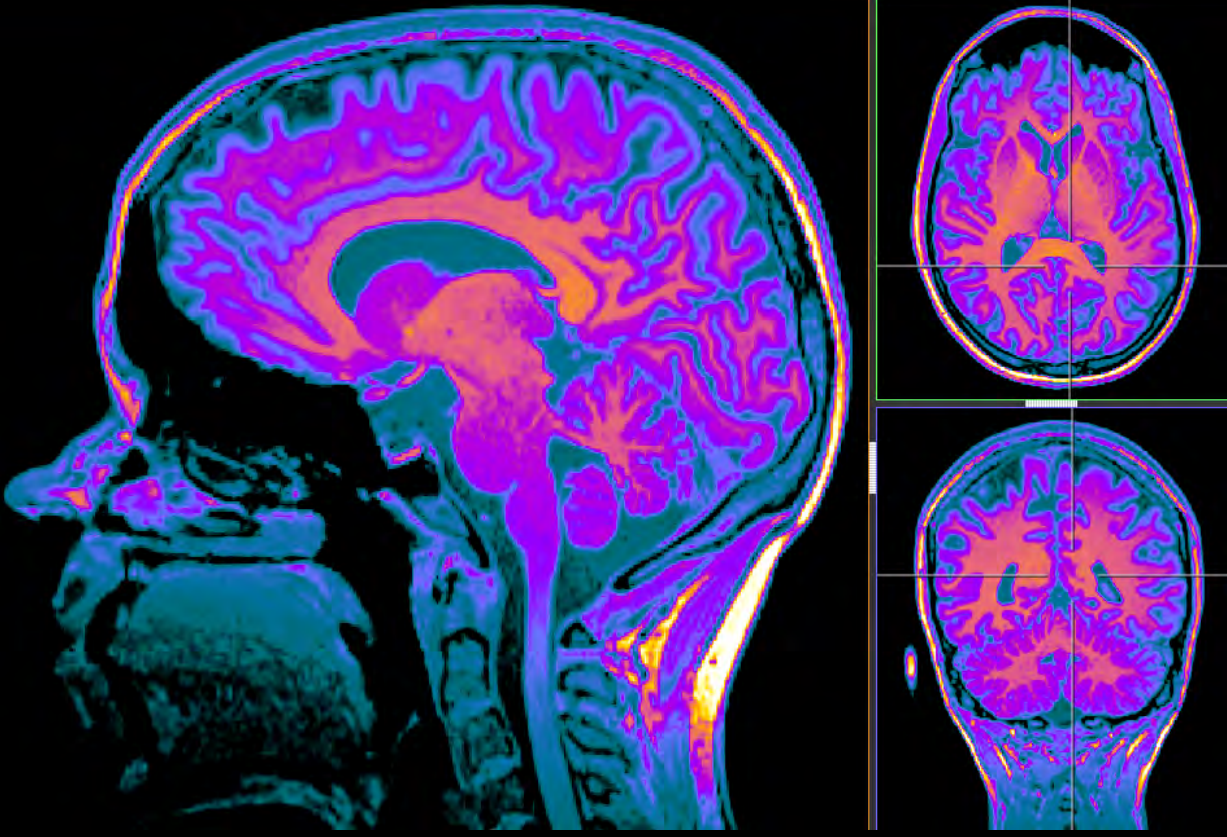

DEVELOPMENT OF A REAL-TIME FUNCTIONAL MAGNETIC RESONANCE IMAGING PIPELINE

Internship report

by

Adnan Muhammad Niazi
University of Twente
s1029789

Internal Supervisors
Marcel van Gerven
Philip van den Broek
Peter Desain
Donders Center for Cognition,
Radboud University Nijmegen

External Supervisor
Mannes Poel
University of Twente

January 2012

CONTENTS IN BRIEF

CONTENTS

ACKNOWLEDGMENTS

I would like to thank Stephan Klanke for providing support in figuring out some bugs. Many thanks to Markus Barth for the many fruitful discussions and for modifying the PACE sequences. Kudos to Paul Gaalman from the MRI Lab for always being so accommodating and for helping me whenever I asked for it. I highly appreciate the support from Philip van den Broek for the countless number of times that he had to modify the stimulus presentation scripts for me. I am particularly grateful to Marcel van Gerven for his time and effort in thoroughly reviewing this document and for his invaluable suggestions. Last but not least, I am grateful to Mannes Poel for his continued supervision through out this process.

GLOSSARY

BrainStream BrainStream is a MATLAB-based software package for realtime processing of continuous data streams.

BOLD BOLD or Blood Oxygenation Level Dependent functional magnetic resonance imaging is a form of magnetic resonance imaging of the brain that registers blood flow to functioning areas of the brain.

Channels A channel in fMRI refers to the time course data of one voxel. A typical fMRI volume contains thousands of these channels.

FieldTrip FieldTrip is the MATLAB software toolbox for EEG, MEG and fMRI analysis that is being developed at the Center for Cognitive Neuroimaging of the Donders Institute for Brain, Cognition and Behavior together with collaborating institutes.

Matrix or Display Matrix The total number of pixels in the selected matrix, which is described by the product of its phase and frequency axis. For example if the readout and phase resolution of an fMRI is 64 and 48 respectively then the display matrix would be 64 x 48.

fMRI fMRI or Functional magnetic resonance imaging, is a technique for measuring brain activity by detecting the changes in blood oxygenation and flow that occur in response to neural activity. When a brain area is more active it consumes more oxygen and to meet this increased demand blood flow increases to the active area. fMRI can be used to produce activation maps showing which brain regions are involved in a particular mental process.

PACE PACE or Prospective Acquisition CorrEction is Siemens implementation of online motion correction in which MRI gradients are changed whenever head motion occurs such that the same spatial location in the brain are scanned no matter where the head moves inside the scanner.

HRF HRF or Hemodynamic response function is the predicted BOLD response to an instantaneous neuronal signal. Its shape and delay varies between individuals, across brain areas, with alertness etc. The sluggishness or inertia of the HRF limits the temporal resolution of fMRI.

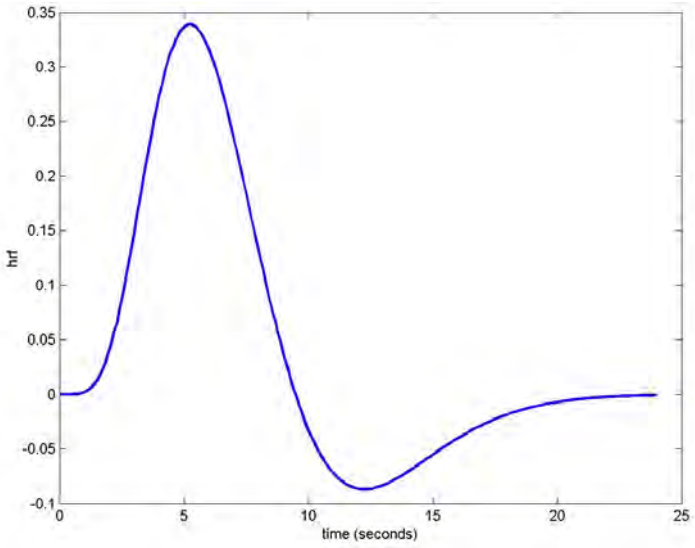

Figure 0.1 Hemodynamic response function of the neural activity evoked by a stimulus at time point 0s. The BOLD activity reaches the peak at around 6s and decays with an undershoot at 12s before reaching a steady state at around 20s. The delay of 6s between the stimulus and getting a maximum BOLD signal is called the hemodynamic inertia

.

Longitudinal equilibrium When a scanner is first started the MR signal is much higher than what it will be later on during the experiment. It takes about at least 3s before a quasi stable MR signal level(equilibrium) is reached. Because the scans

collected till the longitudinal equilibrium has reached have higher contrast then the scans that follow, therefore these few scans need to discarded from analysis.

Preprocessing data buffer The preprocessed data buffer is a FieldTrip buffer which receives data from raw data buffer, one sample at a time, and applies all the preprocessing steps to it. The preprocessed data is stored so that another application down the line can access it whenever needed.

Phase resolution The two axes in an image are thus the "readout" axis and the "phase-encode" axis. For a transaxial image in the traditional orientation (z-axis along the main magnetic field), the z-gradient is used to select the slice. The y-gradient may be used for "phase-encoding" and the x-gradient for "frequency-encoding" or "readout" (or vice versa). The number of voxels in the y direction, is known as the phase resolution.

Readout resolution The two axes in an image are thus the "readout" axis and the "phase-encode" axis. For a transaxial image in the traditional orientation (z-axis along the main magnetic field), the z-gradient is used to select the slice. The y-gradient may be used for "phase-encoding" and the x-gradient for "frequency-encoding" or "readout" (or vice versa). The number of voxels in the x direction, is known as the readout resolution.

Raw data buffer Raw data buffer is a FieldTrip buffer that can receive raw neurological data from a data acquisition client such EEG, MEG or fMRI. The FieldTrip buffer is a network transparent TCP server that allows the acquisition client to stream data to it per sample or in small blocks, while at the same time previous data can be analyzed.

Real-time In streaming applications, an operation on a data sample is said to be real-time if it is completed before the next sample becomes available.

Repetition time or TR TR is the time interval between two successive scans. One scan is acquired in each TR. Each scan contains N slices as specified in the acquisition protocol.

StimBox A toolbox (in progress) which uses Psychtoolbox for developing dynamically updateable agenda-based stimulus presentation designs.

Scan A scan refers to the one acquisition of the whole/ partial brain data. Each scan contains N different slices as specified in the fMRI scanner setting.

Sample A sample in real-time fMRI refers to one volume of MRI brain data.

SPM SPM is a MATLAB software package implementing Statistical Parametric Mapping for neuroimaging data. Statistical Parametric Mapping refers to the construction and assessment of spatially extended statistical processes used to test hypotheses about functional imaging data.

Slice A slice is one slab of brain voxels. Each slice contain R x P voxels, where R is the readout resolution and P in the phase resolution. N slices of the brain constitute one volume or scan.

Volume A volume is the same as a Scan. Each volume contains N slices as specified in the MRI sequence. In normal 2D-EPI sequences, each volume is acquired slice by slice. In 3D EPI sequences, all the slices are acquired at once.

CHAPTER 1

INTRODUCTION

1.1 FUNCTIONAL MAGNETIC RESONANCE IMAGING (fMRI)

Magnetic resonance imaging(MRI) is a non-invasive technique that uses strong magnetic fields to form high resolution images of the brain and body. *Functional* magnetic resonance imaging or fMRI is an extension of MRI to measure quick and tiny metabolic changes that take place in the active brain. Thus, fMRI studies are capable of providing not only an anatomical view of the brain, but a minute-to-minute recording of actual brain activity based on blood-oxygen-level-dependent (BOLD) signal changes related to neuronal activity across the entire brain. The ability of fMRI to track changes in the neuronal activity has proved very useful in localizing task related brain activations and hence in the functional mapping of brain.

1.2 REAL-TIME FUNCTIONAL MAGNETIC RESONANCE IMAGING (rtfMRI)

In typical fMRI studies, data acquired during an experiment, is analyzed offline which usually take several hours and sometimes even several days after the initial acquisition of the data. This presents some unique problems. For example,

- The scanner operator may not be able to detect technical failures or a subject's poor compliance on time, and will not be able to react in timely manner [14]. Later on, if a problem is discovered during the offline analysis, then often there is very little that could be done to alleviate it. This may result in discarding the whole session data. Since a typical scanner costs around 600 Euros per hour to operate, such a loss of data could be very costly. To avoid this, the ideal scenario would be to know the results while the subject is still inside the MR scanner and before the experiment is concluded.
- In a traditional fMRI experiment, a subject cannot be presented with any kind of neurofeedback because the processing of fMRI data is done at a later time and no results are available to be presented to the subjects while the cognitive task is being performed inside the scanner. If this processing can be done in real time, then the subject's task performance can be fed back to them help them modulate their brain activation patterns.

To overcome all these problems real-time fMRI was introduced. In a real-time fMRI experiment, the data is processed as soon as it becomes available. In an fMRI experiment, a new scan (brain volume) becomes available every TR (Repetition Time) seconds. For real-time studies, the TR may range from 1s to 2.5s, depending on how many brain slices are acquired. Therefore for real time operation, its crucial that each scan be processed within a TR, otherwise the data processing will lag behind the data acquisition and the setup will no longer remain real time. With the advent of faster hardware and optimized software routines, real time analysis and acquisition of the fMRI has become possible.

1.2.1 Applications of real-time functional Magnetic Resonance Imaging

Real time analysis of fMRI data can be used to:

- Monitor data quality and changing scan parameters, if need be
- Monitor changes in subject's attention and performance
- Assess experiment success while its being performed
- Retrieve rapid results resulting in cost saving

- Novel paradigm designs with feedback

Given below are some of the domain specific uses of real-time.

Controlling perception of pain Several studies have shown converging evidence the real-time fRMI neurofeedback can help subjects to regulate their own brain activity in specific regions [13, 5, 11]. This renders real time fMRI useful for clinical applications as well. If an individual can learn to directly control activation of localized regions within the brain, this approach might provide control over the neurophysiological mechanisms that mediate behavior and cognition and could potentially provide a different route for treating disease. Subjects can be trained to learn to control activation in regions involved in pain perception and regulation [6].

Pre-surgical planning Other application of real-time fMRI could be in pre-surgical planning. For instance, a surgeon operating a lesion could use functional brain studies to minimize the extent of the damage that could result in the operation [7]. Real-time results of these functional studies are therefore critical. In a clinical setting where a doctor has to see many patients everyday, the real-time analysis of the patient data not only saves cost but also paves the path for faster treatment process.

Observing brain disorders in action In spreading disorders such as Jacksonian seizures or migraine, the possibility of observing activation maps of such phenomena in real time could lead to a better understanding of the spreading mechanism of the disorder as well as to the development of therapeutic interventions to arrest the symptoms progression [9].

These and several other potential applications are the compelling factors that motivate us to develop a fully real-time analysis system for fMRI time series. In simple terms, the research's goal is to see the activation map unfolds in real time as the subject performs the designated task, thereby enabling us to look how the brain works in real time.

1.2.2 Challenges involved in real-time fMRI

In order to design a real-time fMRI pipeline, we need to overcome two main difficulties, namely:

1. the need for a general real-time analysis tool and
2. the computational requirement.

The former requires analysis tools that we can use in real-time to be able to process fMRI data in the same rigor as that of offline analysis methods, while the latter demands computational prowess that can cope with the computational demands.

1.2.3 Project Goals

Although the groundwork for this project was laid by Stephan Klanke in 2010, the pipeline he developed worked only for the 1.5T scanner. Since 3T field strength scanners are now the de facto standard for carrying out neurological investigations, there was a need to develop and test the pipeline on 3T scanners. The current work therefore, has the following two aims:

- Export the real-time pipeline to the 3T scanner
- To develop real-time brain extraction so that non brain voxels, such as skull and Cerebral Spinal Fluid (CSF) voxels are removed so that only voxels constituting the gray and white matter remain.
- To Develop real time slice timing correction using Sinc interpolation which is an improvement over the previously developed linear interpolation based slice timing correction.
- To modify the pipeline to cope with PACE (Prospective Acquisition CorrEction) sequences. PACE based sequences spit out two series of images only one of which needs to be used in any experiment. The modified pipeline should allow not only the option of using PACE but should also allow the user select which of the two series is used for analysis.
- To test and develop the current protocol for starting an fMRI experiment and identify the bottlenecks in the design and give recommendations for future improvements.

The real-time fMRI pipeline available at the time was scantly documented and there was a need for a detailed document which is thorough enough for a successful experiment, yet easy enough to comprehend and follow. This report of this project is therefore meant to serve as a formal training manual for prospective users aiming to develop their own real time fMRI experiments using this pipeline. The report merges the documentation currently available on the FieldTrip website and extends this information to give a comprehensive guide to real-time fMRI experiment design and execution.

The rest of this report gives a detailed description of the entire real-time setup and its various components. Chapter 2 gives a systems level overview of the whole experimental setup required for running a real time experiment on Siemens 3T scanners. The fours chapters that follow explain each of the four modules of the real time setup. The third to last chapter mentions step-by-step protocol instructions on how to start the fMRI pipeline for a successful experiment execution. The second to last chapter mentions the results of the evaluation of the pipeline. The last chapter gives suggestions for future improvements.

CHAPTER 2

OVERVIEW OF THE REAL-TIME fMRI PIPELINE

The current implementation of the real-time pipeline contains four separate module as shown in Figure 2.1, each one performing some specific task in the pipeline. These modules are:

- Scanner Module
- Preprocessing Module
- BCI Module
- Synchronization Module

To better understand what each module does in the pipeline, a brief description of each of these modules will be presented in this chapter (A more detailed description of the each of these modules can be found in the subsequent chapters).

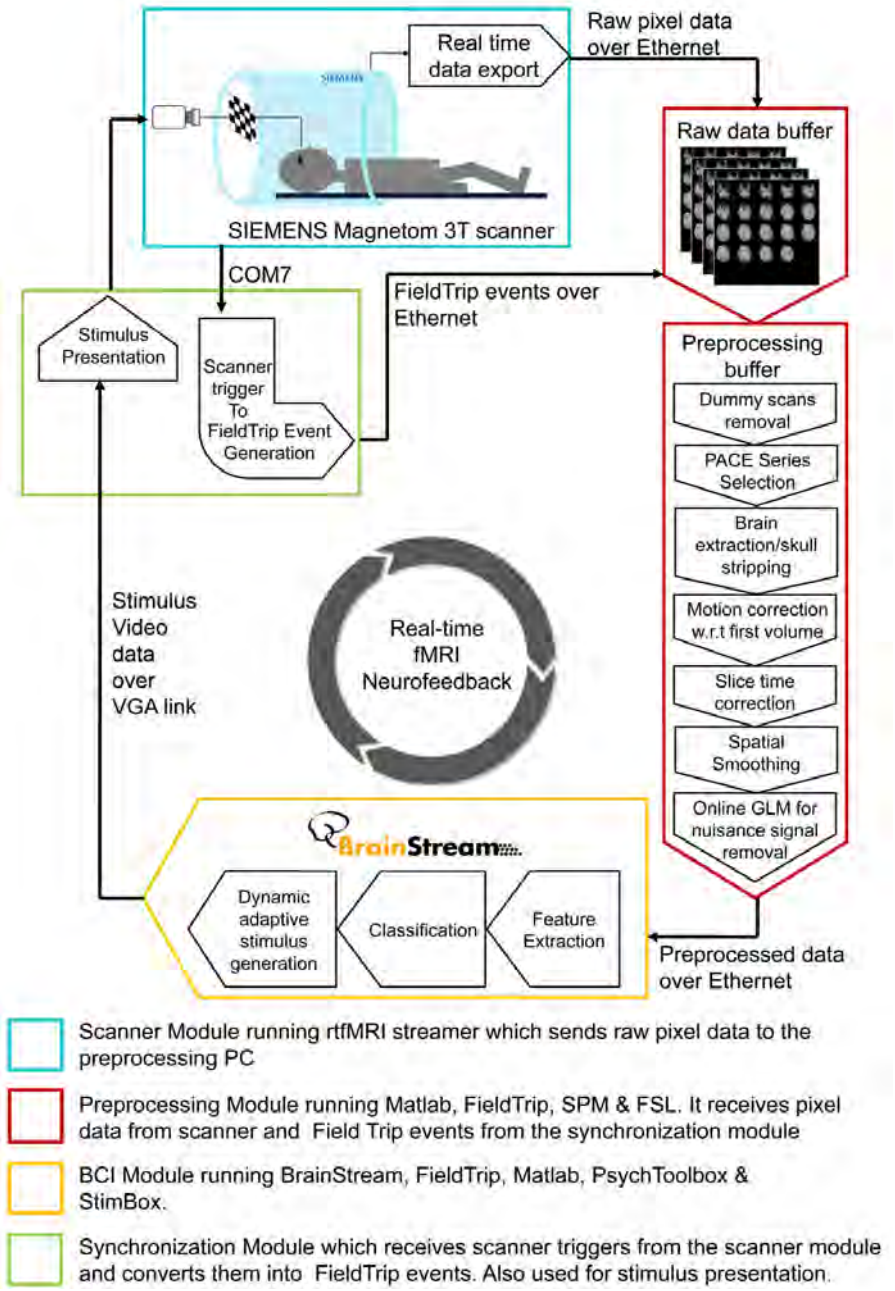

Figure 2.1 Real-time fMRI pipeline at Donders Institute for SIEMENS 3T Magnetom Scanner

2.1 SCANNER MODULE

The scanner module contains the scanner monitor PC where scan paraments are adjusted at the start of each rtfMRI experiment. This is the only computer in the pipeline that can directly control the operation of the scanner. No other PC in the pipeline is allowed to control the scanner.

Furthermore, on the scanner monitor PC, a program called gui_streamer is run which sends each newly acquired functional scan to the raw data buffer in the preprocessing module, where the raw data is received and sent further down the preprocessing pipeline.

The scanner monitor PC also sends scanner triggers to the synchronization module where the scanner triggers are converted into FieldTrip buffer events, and sent to the FieldTrip buffer in the preprocessing module where these events trigger the acquisition of the scan data from the scanner module.

2.2 PREPROCESSING MODULE

The preprocessing module is a computer on the network that receives the raw scans from the scanner module and scanner trigger events from the FieldTrip Event Generation module. The received FieldTrip events trigger the acquisition of a new volume from the scanner module.

The data received in the raw data buffer is passed into a second buffer where the fMRI data is preprocessed one scan at a time. The preprocessing steps include dummy scan rejection, motion correction, slice time correction, brain extraction, spatial smoothing and online GLM for nuisance signal removal.

The preprocessed data from this buffer is then passed on to the BCI module.

2.3 BCI MODULE

The BCI module has the responsibility to receive the preprocessed streaming data from the preprocessing module and use it in a real time BCI loop. In our setup, we used BrainStream toolbox to control the BCI loop. This module has various responsibilities as mentioned below:

- read streaming data from the preprocessing module
- extracting features
- classifying features

- generating stimulus based on the result of the classification (dynamic adaptive stimulus generation)

The stimulus generated in this module is sent to the Synchronization module. This is because in our setup, this computer is connected to the scanner projector.

2.4 SYNCHRONIZATION MODULE

Whenever a new functional scan is started the scanner generates a trigger. This trigger can be used to signal the raw data buffer (FieldTrip) on the preprocessing module to start acquiring the scan data. However, this trigger cannot be used directly by the FieldTrip buffer. It has to be converted into a format acceptable to the buffer. This is what happens in synchronization module. The scanner trigger received on the COM7 port of this module is converted into FieldTrip buffer events and passed to the raw data buffer on the preprocessing module to trigger data acquisition whenever a new functional scan data becomes available.

The module also receives a ‘RESET’ message on its UDP port 1990 which resets the internal pulse counter. Its very important that before the start of the functional scans a ’RESET’ is sent by the scanner.

Because this is the only computer connected to the scanner projector, the stimulus (if generated on some other PC, which in our case it is) should be connected to this module.

CHAPTER 3

SCANNER MODULE

This module has a Windows PC that runs the Siemens software which controls the operation of the entire scanner. Subjects can be registered and scan parameters adjusted at the start of the experiment on this PC. Furthermore, the localizer images acquired during the initial localizer scan can also be viewed on this computer. These localizer scans, or Haste scans as they are called, are used in positioning the slices over brain regions of interest for the subsequent functional runs.

Once the functional sequence is run, the data generated needs to be transferred to a PC on the network so that it can be further processed. Although Siemens software (Version VB17) provides the facility for real-time export of the fMRI images, there seems to be considerable jitter in the arrival of the files using this tool. Furthermore, this tool does not allow for the real-time export of the images to PC on the network (see Figure 3.1). Due to these limitations, it was decided to design a custom real time export tool that could export the raw pixel data to a remote PC on the network for further processing.

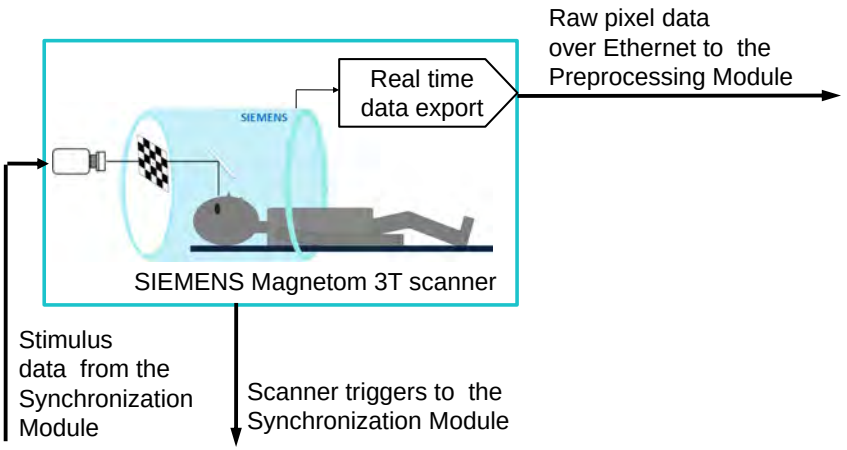

Figure 3.1 Block diagram of the Scanner Module with it's various IO's

3.1 PIXEL DATA/ MOSAIC FILES

The scanner stores raw pixel data belonging to each functional scan in the form a Mosaic file. The Mosaic format stores the 3D image slices as a 2D grid - or mosaic. With the current Siemens scanner software (VB17A), a new mosaic file `E:\IMAGE\xx-yyyy\zzzzzzz.PixelData` is created on the `E:\` drive of the host computer immediately after each scan. This file contains pixel data as unsigned 16-bit integers, where different slices show up as tiles of a mosaic. The mosaic seems to be always square, and blank tiles are appended if the number of slices is smaller than the number of tiles in the mosaic.

■ EXAMPLE 3.1

The MR sequence is set up to scan N=35 slices with readout resolution R=64 pixels and phase resolution P=48 pixels. In this case, the mosaic will contain 6x6 tiles (as shown in Figure 3.2) with 1 empty tile marked by `--` and slices ordered as follows:

```
01  02  03  04  05  06
07  08  09  10  11  12
13  14  15  16  17  18
19  20  21  22  23  24
25  26  27  28  29  30
31  32  33  34  35  --
```

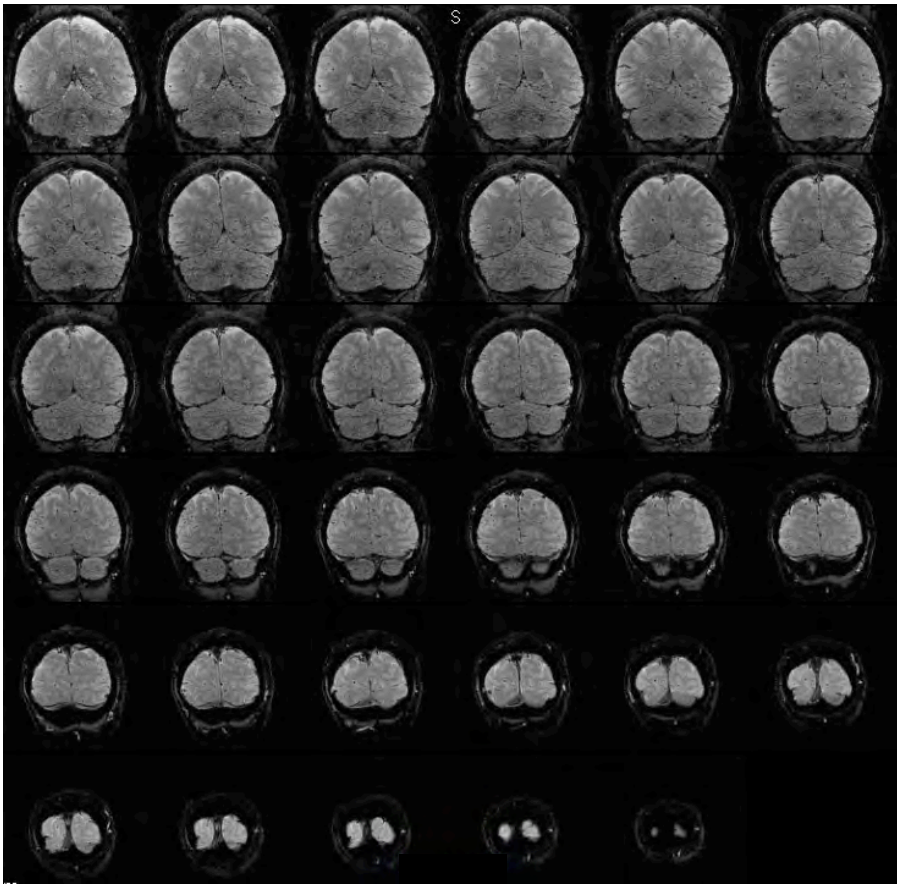

Figure 3.2 Mosaic image of functional data when 35 slices are acquired in each scan. The mosaic contains 6 x 6 tiles and the last tile is empty

The pixel dimensions of the mosaic will be (64*6) x (48*6), that is, 384 x 288, and thus the total number of pixels is 110592, corresponding to a file size of 221184 bytes. Within the file, the pixels are written row after row, that is, the

first 768 bytes contain the 384 pixels of the first row, corresponding to the first rows of slices 01-06, and so on.

Within the FieldTrip buffer, each scan is represented as one sample with RxPxN channels, with data ordering as in MATLAB®, that is, the pixel data is reshaped such that slices (and their rows) are contiguous in memory, and empty tiles are dropped. For the above example, we would have 64x48x35 = 107520 *channels*. The data format is kept as `INT16_T`.

EXAMPLE 3.2

The MR sequence is set up to scan N=19 slices with readout resolution R=64 pixels and phase resolution P=64 pixels. In this case, the mosaic will contain 5x5 tiles (as shown in Figure 3.3) with 6 empty tiles marked by -- and slices ordered as follows:

```
01   02   03   04   05
06   07   08   09   10
11   12   13   14   15
16   17   18   18   --
--   --   --   --   --
```

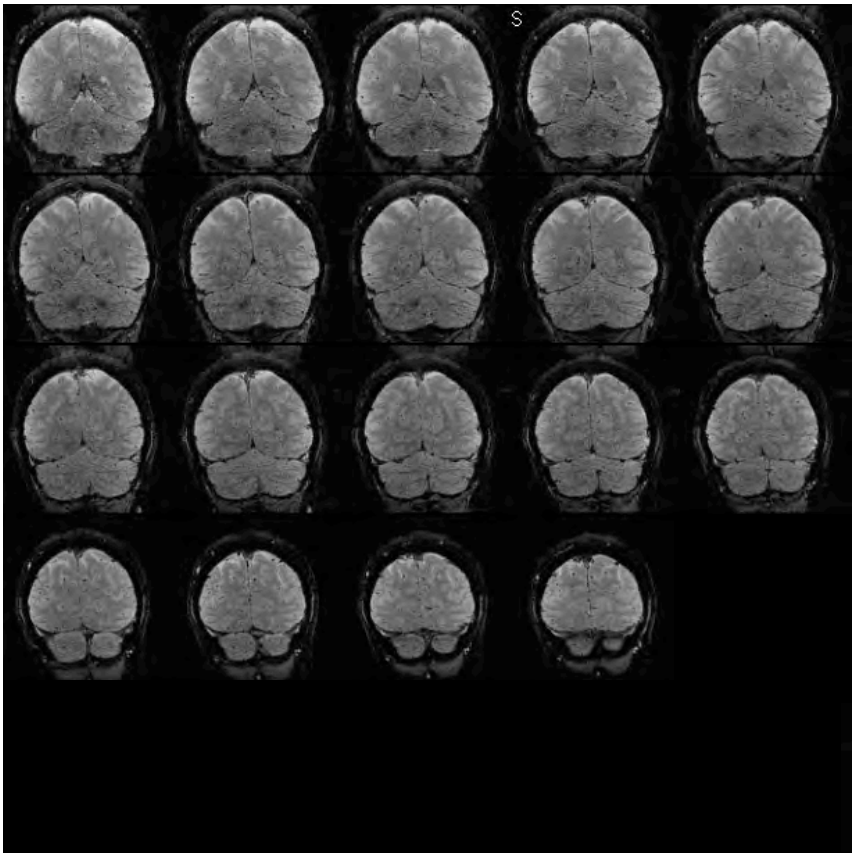

Figure 3.3 Mosaic image of functional data when 19 slices are acquired in each scan. The mosaic contains 5 x 5 tiles and the last 6 tiles are empty

The pixel dimensions of the mosaic will be (64*5) x (64*5), that is, 320 x 320, and thus the total number of pixels is 102400, corresponding to a file size of 204800 bytes. Within the file, the pixels are written row after row, that is, the first 640 bytes contain the 320 pixels of the first row, corresponding to the first rows of slices 01-06, and so on.

Within the FieldTrip buffer, each scan is represented as one sample with RxPxN channels, with data ordering as in MATLAB®, that is, the pixel data is reshaped such that slices (and their rows) are contiguous in memory, and empty tiles are dropped. For the above example, we would have 64x64x19 = 77824 *channels*. The data format is kept as `INT16_T`.

3.2 PIXELDATAGRABBER

The next problem is to grab these mosaic files, as they are generated and transfer them to a remote PC on the network. In order to react efficiently to a new mosaic file, which shows up as a file with name and location not known in advance (apart from its suffix), the Windows API function `ReadDirectoryChangesW` is used to monitor `E:\IMAGE` and all of its subdirectories. Whenever a new file is created or modified anywhere in that tree, a Windows event is triggered and the corresponding path is made available. This mechanism is wrapped up in the C++ class `FolderWatcher`.

A second C++ class, `PixelDataGrabber`, encapsulates the actual real-time fMRI acquisition mechanism based on the FolderWatcher and client-side code of the FieldTrip buffer. Detailed Doxygen-style documentation is provided in `PixelDataGrabber.h`, and developers can also look at `pixeldata_to_remote_buffer.cc` for a simple example of using this class in a command-line application. A slightly more complex program is compiled from `gui_streamer.cc`, which combines the `PixelDataGrabber` with a small GUI written with FLTK (`http://www.fltk.org`). This program provides a few buttons for starting and stopping to monitor for new files, and to connect to/disconnect from a FieldTrip buffer (see Figure 3.4).

3.3 PROTOCOL INFORMATION

How does the `PixelDataGrabber` determine the number of slices and their dimensions? For this to work, the best way is to modify the MR sequence by adding

```
#ifndef VXWORKS
   pMrProt->fwrite("E:\\image\\mrprot.txt")
#endif
```

to the function `fSeqCheck` in the sequence code, which is executed once before the first scan. This will dump the complete protocol information to the specified location.

With the `PixelDataGrabber` listening for files in `E:\image`, it will note this and immediately parse the new protocol. The information written to that file is the same that is contained in one of the private tags of the DICOM headers, which the scanner writes using the normal (offline) mechanisms. The filename `mrprot.txt` is currently hard-coded in both the `PixelDataGrabber` and the standard sequences used at the DCCN. If you run an unmodified MR sequence that does not dump the information,

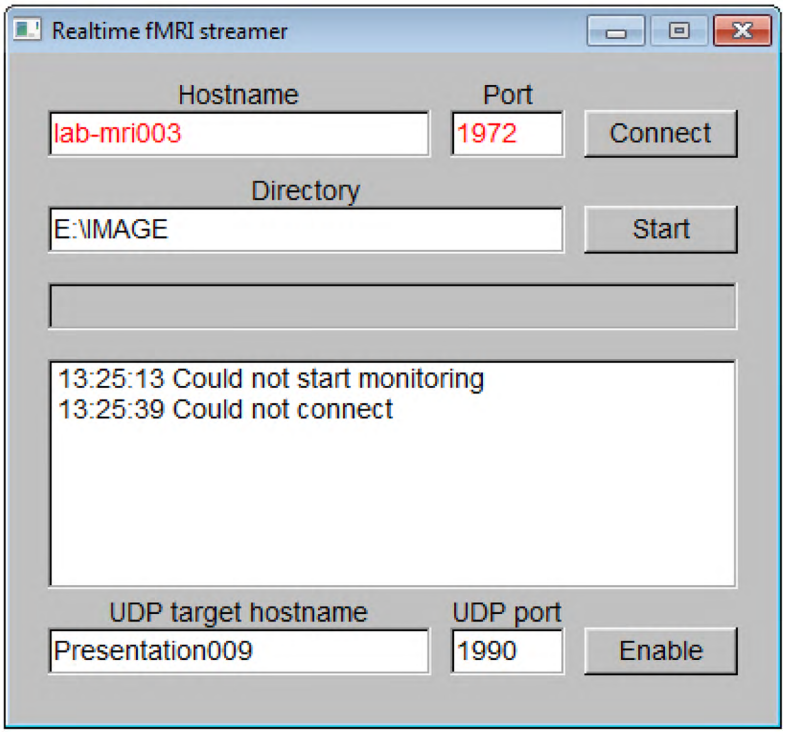

Figure 3.4 gui_streamer: A graphical user interface to select the target for real-time fMRI data transfer

you can try to create your own mrprot.txt and place it in `E:\image` before running the scans. In this case, the `PixelDataGrabber` will read that file when the first scan arrives. In case the `PixelDataGrabber` encounters a mismatch between protocol specifications and the size of the .PixelData files, it will report an error and not write the sample. The most important ingredients for a hand-made protocol file are shown in the following example:

```
alTR[0]                                 = 1500000
lContrasts                              = 1
sKSpace.lBaseResolution                 = 64
```

```
sSliceArray.lSize                          = 19
sGroupArray.asGroup[0].dDistFact           = 0.1
sSliceArray.ucMode                         = 0x1
ucReconstructionMode                       = 0x1
sSliceArray.asSlice[0].sPosition.dSag      = 0.2182624061
sSliceArray.asSlice[0].sPosition.dCor      = -11.23376707
sSliceArray.asSlice[0].sPosition.dTra      = -7.52395635
sSliceArray.asSlice[0].sNormal.dSag        = 0.0004806160937
sSliceArray.asSlice[0].sNormal.dCor        = 0.3056497638
sSliceArray.asSlice[0].sNormal.dTra        = 0.9521438919
sSliceArray.asSlice[0].dThickness          = 3
sSliceArray.asSlice[0].dPhaseFOV           = 211
sSliceArray.asSlice[0].dReadoutFOV         = 211
sSliceArray.asSlice[0].dInPlaneRot         = 0.9534438919
```

Given below is the description of each of the field so that you can make your own mrprot.txt file based on your protocol settings. This information would be useful if you are unable to modify the scanner sequence to dump the protocol information in the mrprot.txt file at the start of the functional run. In this case, you can use the information below to make your own mrprot.txt file.

3.3.1 alTR[0]

Specify the Repetition Time (TR) in microseconds in this field. If the TR is 1.5s, then set

```
alTR[0]                                    = 1500000
```

3.3.2 lContrasts

Specify the number of echoes in the acquisition protocol in this filed. If the number of echoes is 1, then set

```
lContrasts                                 = 1
```

If the number of echoes is 3, then set

```
lContrasts                                 = 3
```

3.3.3 sKSpace.lBaseResolution

Specify the matrix size in this field. If the matrix size is 64 x 64, then set

```
sKSpace.lBaseResolution                    = 64
```

3.3.4 sSliceArray.lSize

Specify the number of slices acquired in each TR in this field. For example, if the number of slice acquired in each scan is 19 then set

```
sSliceArray.lSize                    = 19
```

3.3.5 sGroupArray.asGroup[0].dDistFact

Specify the distance factor between the slices in this field. If the distance factor is 10 percent of the slice thickness then set

```
sGroupArray.asGroup[0].dDistFact     = 0.1
```

If the slices have no gap between them, then set

```
sGroupArray.asGroup[0].dDistFact     = 0.0
```

3.3.6 sSliceArray.ucMode

Specify the slice order in this field. If the slices are acquired in ascending order, then set

```
sSliceArray.ucMode                   = 0x1
```

If the slices are acquired in descending order, then set

```
sSliceArray.ucMode                   = 0x2
```

If the slices are acquired in interleaved order, then set

```
sSliceArray.ucMode                   = 0x4
```

3.3.7 ucReconstructionMode

Different sequences can output different image types. This field specifies the type of image generated and whether to keep the magnitude or phase part of the image or both. For example, to select single magnitude image, set

```
ucReconstructionMode                 = 0x1
```

For usual EPI sequences used in real time experiments, this is the recommended setting.
To select single phase image, set

```
ucReconstructionMode                 = 0x2
```

To select real part only (single image), set

```
ucReconstructionMode                    = 0x4
```

To select magnitude+phase image (we need to skip the phase), set

```
ucReconstructionMode                    = 0x8
```

To select real part+phase (we need to skip the phase), set

```
ucReconstructionMode                    = 0x10
```

For PSIR(Phase Sensitive Inversion Recovery Sequence), set

```
ucReconstructionMode                    = 0x20
```

3.3.8 sSliceArray.asSlice[0].sPosition

The three fields mentioned below should encode slice position (X/Y/Z) of the first slice.

```
sSliceArray.asSlice[0].sPosition.dSag    = 0.2182624061
sSliceArray.asSlice[0].sPosition.dCor    = -11.23376707
sSliceArray.asSlice[0].sPosition.dTra    = -7.52395635
```

3.3.9 sSliceArray.asSlice[0].sNormal

The three fields mentioned below should encode the slice normal vector of the first slice.

```
sSliceArray.asSlice[0].sNormal.dSag      = 0.0004806160937
sSliceArray.asSlice[0].sNormal.dCor      = 0.3056497638
sSliceArray.asSlice[0].sNormal.dTra      = 0.9521438919
```

3.3.10 sSliceArray.asSlice[0].dThickness

Specify slice thickness in the field. If the slice thickness is 3mm, then set

```
sSliceArray.asSlice[0].dThickness        = 3
```

3.3.11 sSliceArray.asSlice[0].dPhaseFOV

Specify the Field of View (in mm) in the phase encoding direction in this field.

```
sSliceArray.asSlice[0].dPhaseFOV         = 211
```

3.3.12 sSliceArray.asSlice[0].dReadoutFOV

Specify the Field of View (in mm) in the frequency encoding direction in this field.

```
sSliceArray.asSlice[0].dReadoutFOV        = 211
```

3.3.13 sSliceArray.asSlice[0].dInPlaneRot

In this field, specify in-plane-rotation of the first slice around normal.

```
sSliceArray.asSlice[0].dInPlaneRot        = 0.9534438919
```

If no such information is available then remove this field from the mrprot.txt file.

3.4 DECODING THE PROTOCOL FILE

`Siemensap`, a plain C library in `fieldtrip/realtime/datasource/siemens` provides some functions and data types to parse the ASCII format Siemens protocol data into a list or tree of key/value items. Currently supported value types are strings, long integers, and double precision numbers. Field types are determined automatically to a large extend (e.g. a dot in a number implies a double precision value), but some special rules are added. For example, a field name that starts with 'd' will always be parsed as a double precision value, even if the value given in ASCII form looks like an integer. Please see siemensap.h for Doxygen-style documentation of the API.

The same C library is also used within the `sap2matlab` MEX file for decoding the ASCII protocol into a MATLAB data structure. However, this gets automatically called in `ft_read_header`, so users won't need to worry if they stick to the usual FieldTrip functions.

CHAPTER 4

PREPROCESSING MODULE

The data acquired on the scanner monitor is transferred in real time to the preprocessing module. The preprocessing module is a computer on the network which receives the raw scans from the scanner module and FieldTrip events from the Synchronization Module. The received FieldTrip events trigger the acquisition of the new volume from the scanner module. This module runs two FieldTrip buffers. The first buffer, called the *raw data buffer*, receives the real time fMRI scans and FieldTrip event at a specified port (1972 by default, but it can changed in the GUI_streamer in Hostname *Port* field and in the serial_event.conf file). The second buffer, called the *processed data buffer*, takes the raw data one scan at a time from the raw data buffer, applies all the preprocessing steps and sends the preprocessed scan to the Classification Module. Furthermore, both raw and preprocessed data streams can be stored at any desired location (network or the local PC). These files can then be rerun offline to simulate the online experiment.

The following operations are performed in the preprocessing pipeline:

- Dummy scan rejection

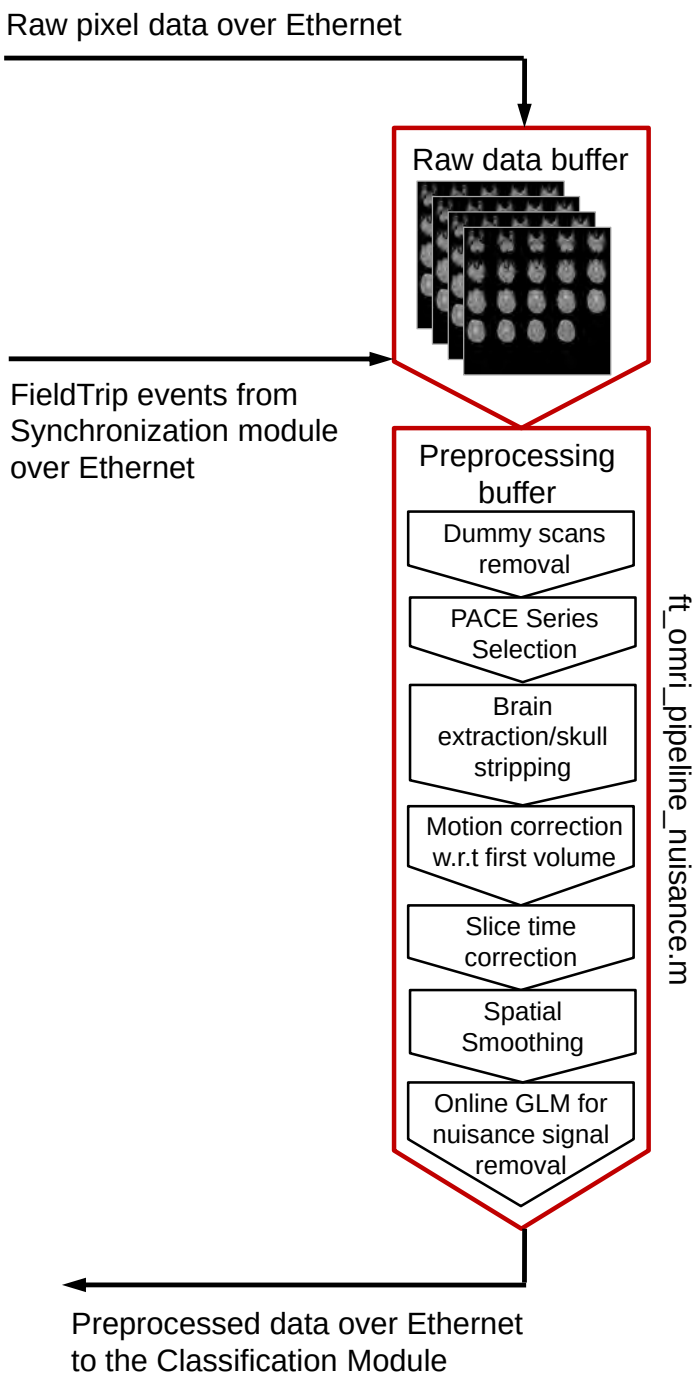

Figure 4.1 Block diagram of the Proprocessing Module with it's various IO's

- Slice time correction
- Motion correction
- PACE series selection
- Brain extraction
- Online GLM for nuisance removal
- Spatial smoothing

All these operations are performed by a FieldTrip function ft_omri_pipeline_nuisance.m. Different configuration setting can be made in this function. We will now describe these different configuration setting for all the preprocessing operations.

4.1 DUMMY SCAN REJECTION

Whenever an fMRI experiment is started, the first few scans have a higher contrast than the scan acquired after longitudinal equilibrium has been achieved. Particularly images with TR<2s have low contrast between grey matter, white matter and CSF, and image registration can be poor. If these first few scans are acquired then they should be dropped afterwards from the analysis. The number of initial scans that should be dropped, depends on the TR and is chosen to guarantee more than 3 seconds of time for stabilization. So;

```
TR > = 3001ms                    1 dummy scan
3001 < TR <= 1501ms              2 dummy scans
1501 < TR <= 1001ms              3 dummy scans
Dummy scans = ROUNDUP(3001/TR)
```

So, in general the number of dummy scan to be dropped can be calculated by:

```
Dummy scans = ROUNDUP(3001/TR)
```

During data analysis, care should be taken to take into account that while dummy scans are being dropped, the scanner triggers are being generated but no actual data is available to be analyzed. For example, if 1000 functional scans are generated by the scanner out of which 5 initial dummy scans are dropped, then there will be only 995 functional scans but 1000 scanner events. The first five dummy scans that are dropped will belong to the first trial (in a block design). Dropping these dummy scans will render the first trial of the cognitive task useless. Thus dropping Dummy scans is not the smartest way to tackle the longitudinal equilibrium issue, however its included in the preprocessing pipeline for the sake of completeness.

The recommended option to tackle longitudinal equilibrium problem is to set Dummy scans to 0 in the preprocessing scrip, and turn ON the *Prep Scans* in the MRI sequence. When the Prep Scans are turned ON, no scans and scanner pulses are generated

until longitudinal equilibrium is reached. When the first scan is generated, the T2 equilibrium has already been reached so there is no need to drop the initial scans. In this case, dummy scans should be set to zero.

Furthermore, motion correction, Online GLM, and brain extraction will only work correctly if either Prep scans are ON and Dummy scans set to zero or if the Prep scan are OFF and Dummy scans are dropped. Both these scenarios are explained with detailed examples below.

EXAMPLE 4.1

In an fMRI experiment, functional scans are acquired at TR=1.5s. The prep scans are turned OFF in the scanner sequence. How many dummy scans should be dropped from the analysis in the preprocessing script?

To answer this question, we use:

```
Dummy scans = ROUNDUP(3001/TR)
            = ROUNDUP(3001/1500)
            = 2
```

Hence in the preprocessing script set,

```
Dummy scans = 2;
```

This scenario is depicted in the Figure 4.2. As you can see the first and the second scan scans have a higher contrast than the subsequent scans. Setting Dummy scans to 2 will get rid of these initial 2 scans.

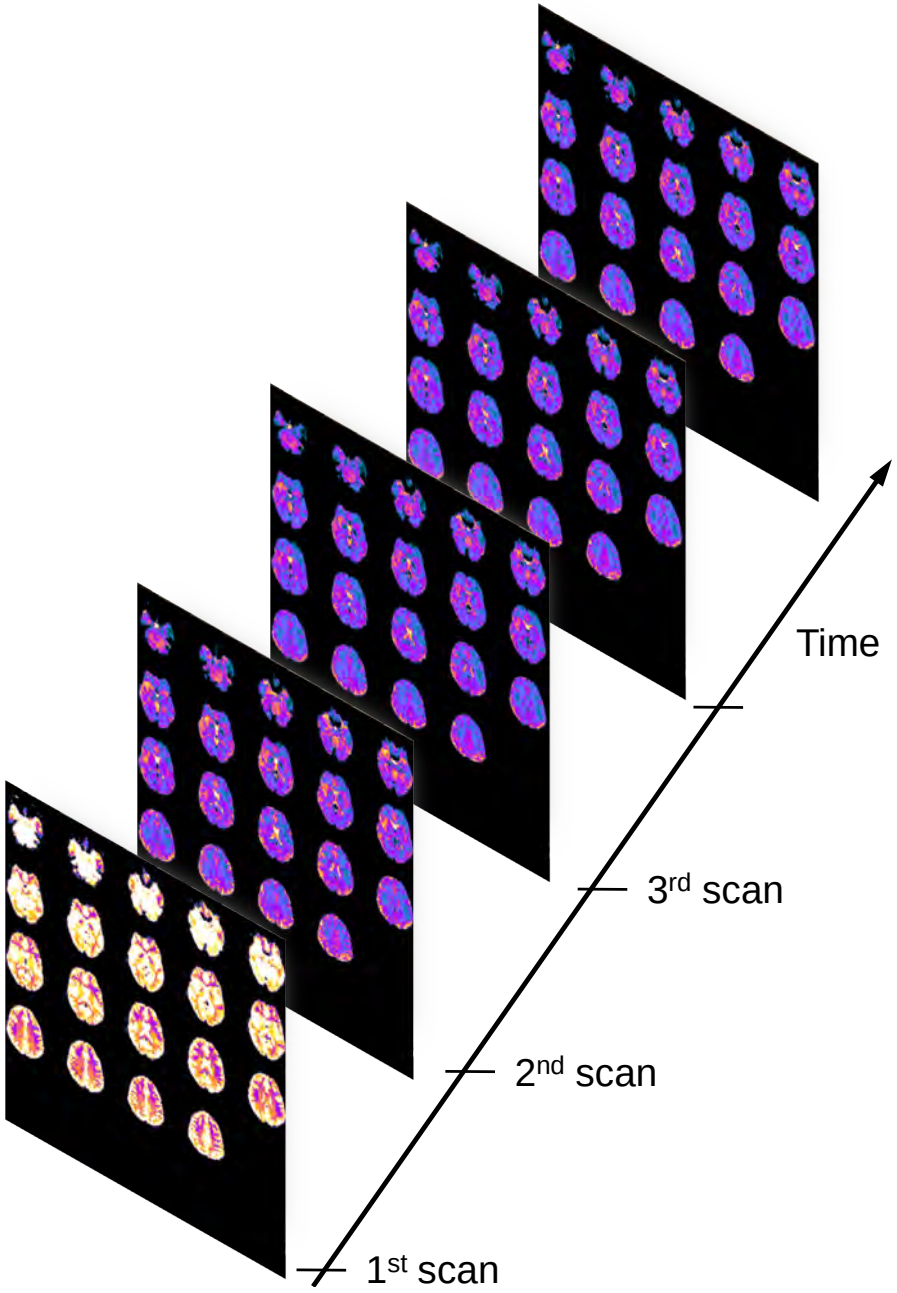

Figure 4.2 The first two scans have a higher contrast than the subsequent scans due to the fact that longitudinal equilibrium has not reached. These initial 2 dummy scans should therefore be dropped.

EXAMPLE 4.2

In an fMRI experiment, functional scans are acquired at `TR=1.5` s. The prep scans are turned ON in the in scanner sequence. How many dummy scans should be dropped from the analysis in the preprocessing script?

Because the prep scans are ON, there is no need to drop any scans because all the scans generated by the scanner will be produced after the longitudinal equilibrium has been reached. So therefore, the following setting should be made in the pre-processing script:

```
Dummy scans = 0;
```

This is demonstrated in the Figure 4.3 . All the scans have the same contrast because when the Prep scan is ON, no scans are generated until the longitudinal equilibrium has been reached. So there is no need to drop any scans.

4.2 SLICE TIMING CORRECTION

The slice timing problem occurs because in fMRI, a brain volume is acquired one slice at a time and the last slice is acquired almost 1 TR later then the first slice. This means that different voxels in the same brain volume, are sampled at different time points on the Hemodynamic Response Function (HRF) curve. Ideally, we would want all the voxels with one brain volume to be sampled at the same time point on the HRF curve. Failure to do so may result in suboptimal statical analysis.

Slice time correction shifts the voxel time courses to align them with sample time of the first slice. So in essence all the voxels are sampled at the same time point. For example, if N=32 slices are acquired in each scan, then slice time correction would align the sample points of all the voxels in slices 2,3,....N=32 to the sample time of slice 1. This changes the data in a way as if the whole volume would have been measured at the same moment in time as the first slice. For slice correction it's imperative that slice ordering be known. This piece of information can be gathered from scanning protocol. If slice ordering is not known, then no slice timing correction should be applied at all.

Slice time correction uses interpolation to align all the voxel sample point to the same location on the HRF curve. Currently, we have implemented two interpolation schemes.

1. Linear interpolation
2. Sinc interpolation

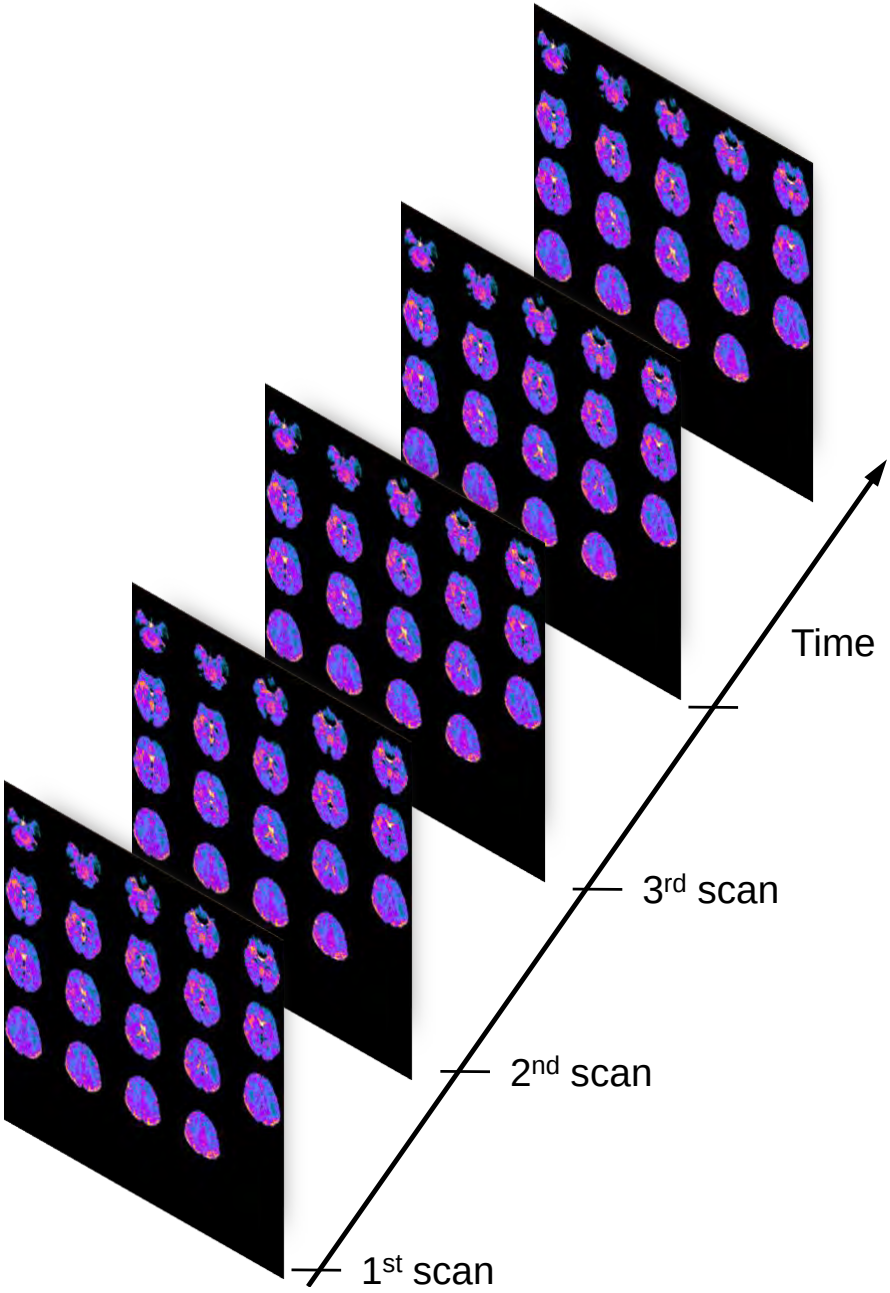

Figure 4.3 All the scans have the same contrast when the Prep scan is ON, no scans are generated until the longitudinal equilibrium has been reached. So there is no need to drop any scan from anaylsis. Dummy scans should be set to zero in this case

Linear interpolation is very fast and can be completed within 0.15 to 0.2 seconds. A more accurate interpolation method is the Sinc interpolation method. Sinc interpolation is slower and may take 1 to 2s depending on the hardware. The Sinc interpolation algorithm is the same as that used by SPM, however it has been modified a bit to run online. Sinc interpolation is accomplished by taking a Fourier transform of the signal at each voxel. The Fourier transform renders any signal as the sum of some collection of scaled and phase-shifted sine waves; once you have the signal in that form, you can simply shift all the sines on a given slice of the brain forward or backward by the appropriate amount to get the appropriate interpolation. There are a couple of pitfalls to this technique, mainly around the beginning and end of your run, but these have largely been accounted for in the current slice time correction implementations of SPM.

4.2.1 When is it a bad idea to use slice time correction?

It's never that bad an idea to use slice time correction, but because at most the signal could be distorted by just one TR, this type of correction isn't as important in block designs. Blocks last for many TRs and figuring out what's happening at any given single TR is generally not a priority, and although the interpolation errors introduced by slice timing correction are generally small, if they're not needed, there's not necessarily a point to introducing them.

4.2.2 FieldTrip functions for slice time correction

Linear Interpolation

```
ft_omri_slice_time_init.m
ft_omri_slice_time_apply.m
```

Sinc Interpolation

```
ft_omri_slice_time_init_sinc.m
ft_omri_slice_time_apply_sinc.m
```

These functions have not been yet been released in the FieldTrip distribution.

4.3 MOTION CORRECTION

Head motion is a major problem during fMRI data acquisition. No matter how much care is taken to fixate the head, some head motion is bound to occur. The sensitivity of the analysis is determined by the amount of residual noise in the image series, so

movement that is unrelated to the task will add to this noise and reduce the sensitivity. Head movement can happen due to many reasons such as:

- Subject's head starts digging into the cushions over time
- Head motion may occur due to breathing
- In some subjects the back of the skull could be oval shaped. In these subjects the head may move quite a lot during scanning because there is no comfortable stable position for these heads.

Any motion causes the brain voxels to shift. For example if a voxel is in slice 22, then after motion this voxel may shift into the adjacent slice (slice 21 or 23). This presents a huge problem. For example, in scenarios where decoding algorithms are applied to voxel time courses, if the voxels continuously shift their position, then it becomes difficult for decoding algorithms to learn any meaningful pattern of activity because the voxels would not represent activity from one location of the brain but would represent activity from many different regions of the brain. Its therefore, important that each voxel represents the activity from one and only one unique location in the brain through out the whole fMRI session. To accomplish this, motion correction algorithm is applied. The motion correction applies rigid body transformation to align all the brain volumes to a reference volume. In our implementation of motion correction, this reference volume is the very first scan acquired. Motion correction uses interpolation to move all the voxels in all subsequent volumes to their position in the reference volume/scan. Various kinds of interpolation can applied, for example:

1. Nearest neighbor interpolation
2. Trilinear interpolation
3. B-Spine interpolation

The type of interpolation used can specified in the `interp` field in the model specification in `ft_omri_align_init.m` file.

```
model = struct('quality',1,'fwhm',5,'sep',4,'interp',2,...
        'wrap',[0 0 0],'rtm',0,'PW','','lkp',1:6,...
        'mat', eye(4), 'time', 2);


'interp'        0   -   Nearest neighbor interpolation
                1   -   Trilinear interpolation
                2-7 -   Degree of B-Spline Interpolation
```

By default, B-spline interpolation of degree 2 is used.

The *Quality* parameter controls how many voxels are used. Highest quality(1) gives most precise results, whereas lower qualities give faster realignment. The idea is that

some voxels contribute little to the estimation of the realignment parameters. The quality parameter is therefore, involved in selecting the number of voxels that are used.

The rest of the parameters are already set to their optimum values and should only be changed by advanced users.

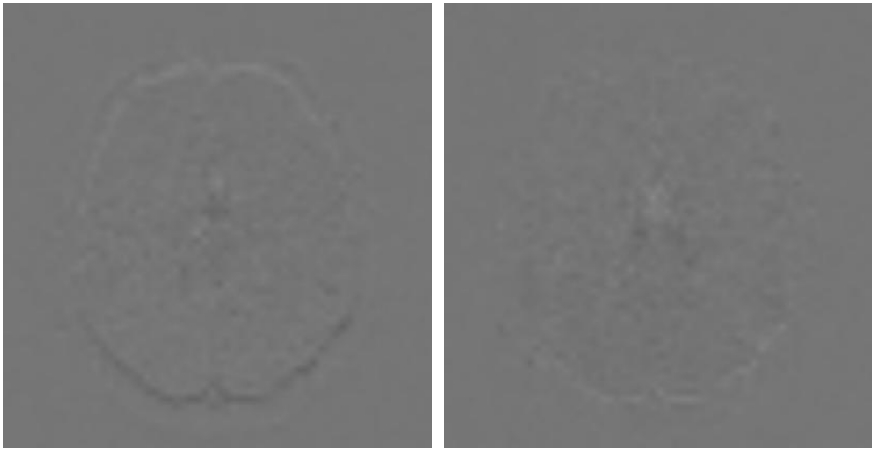

Figure 4.4 (a): Shows the result of subtraction two consecutive scans when no motion correction is applied. The edges are visible. If there is no motion, then nothing should be visible on this plot but as there was motion, therefore we see a lot of spurious activity in this subtraction plot. (b) Shows the result of subtraction of the same two consecutive scans when motion correction is applied. The spurious activity has disappeared to an appreciable extent, which is exactly what we want in a subtraction plot

4.3.1 Limitations of Retrospective Motion Correction

The motion correction that we described above is called *retrospective motion correction*, because it tries to correct for motion, after it has happened. This kind of motion correction has its limitations because it cannot correct for large movements. The quality of fMRI data is strongly hampered in the presence of substantial head movements. Many fMRI experts recommend to reject data sets for further analysis if head motion of more than 1-2 voxels or 5 or more millimeters is detected. Although head motion can be corrected in image space, displacements of the head reduces the homogeneity of the magnetic field, which is fine-tuned or *shimmed* prior to functional scans for a given head position. If head movements are small, retrospective motion

correction is a useful step to improve the data quality for subsequent statistical data analysis.

Gaps between slices can also cause aliasing artifacts. Furthermore, re-sampling can introduce errors, especially tri-linear interpolation. Another limitation of motion correction is that ghosts (and other artifacts) in the images do not move according to the same rigid body rules as the subject.

4.3.2 Limitations of Online Retrospective Motion Correction

Online retrospective motion correction algorithms perform even worse than their offline counterparts and fail if the motion is more than the one slice thickness. In standard offline retrospective motion correction, the motion correction algorithm realigns each volume to one particular reference volume (sometimes the whole procedure is run twice to get a better reference). However, in the online real-time case, we do not have the luxury choosing the best reference to align all the rest of images to. That is why only the first scan is selected as the reference and all the rest of the scans are aligned to this reference scan. In case the necessary translations and rotations to counteract the head motion are large, the realigned volumes will have some "bad" voxels at locations (in reference coordinates) that were outside the scanned volume. These bad voxels are included in a mask so that the experimenter knows how many bad slices are present. The mask does not remain constant but increases through out the experiment as more and more voxels become bad due to motion. This is problematic because if one trains a classifier on good voxels, then after some time through the experiment, some of these voxels may turn bad and hence the classification will severely suffer because the voxels that were used during training, would no longer be there during testing.

An effective but not so elegant fix to this problem is to mask the first few and last few slices before the start of the experiment. The mask should be big enough so as to ensure that throughout the whole experiment, the number of bad voxels would be confined to those few slices which were masked and not used during training and testing. The optimum choice is mask the first 3 slices and last 2 slices.

In the next section we describe a more elegant way of solving this problem.

4.4 PACE SERIES SELECTION

Retrospective motion correction or *correcting for motion after it has occurred*, cannot correct for motion larger than slice thickness. Therefore, Prospective motion correction or *correcting the motion as it occurs*, was introduced by Siemens. PACE or Prospective Acquisition CorrEction [12], is a technique in which the MRI gradients are adjusted if head motion occurs, such that the slice position changes along

with changes in the head position. Thus the slices always remain glued to the same spatial location in the brain no matter where and by how much the brain moves in the scanner.

Whenever motion is detected, PACE recalculates the new gradients and then uses these new gradients for the next scan. Thus, there is a gap of two TRs between the detection of motion and its correction. Any motion that does occur in this two TR interval, cannot be corrected by PACE. This is removed by retrospective motion correction which Siemens calls MoCo. Hence if the PACE sequence is used, it not only corrects the images prospectively but also retrospectively. Therefore, the pace sequence spits out two series of images:

- the first sequence is just PACE (prospectively) corrected
- the second sequence is both PACE (prospectively) and MoCO (retrospectively) corrected.

The experimenter has the choice to decide which series to use in the preprocessing pipeline. We recommend using the second series as it contains much less motion.

4.4.1 Configuring PACE

If the PACE sequence is used then set,

```
cfg.pace = 1;
```

If the pace sequence is not used the set,

```
cfg.pace = 0;
```

If the PACE sequence is used, then an additional choice is to be made as to which series to use. The first series contains PACE corrected images and the second series contains PACE & MoCo corrected images. To select the first series set,

```
cfg.pickSeries = 1;
```

To select the second series set,

```
cfg.pickSeries = 2;
```

4.4.2 Limitations of PACE

PACE cannot correct for motion greater than 25 mm. Furthermore PACE cannot correct any spike motion such as a sneeze. Also any motion below 0.3 mm is also not corrected because its too small a motion to be corrected prospectively. Apart from that, PACE sequences generate twice as much data as a normal no PACE sequence.

4.5 BRAIN EXTRACTION

Brain extraction or *skull stripping* involves removing the skull and cerebral spinal fluid (CSF) voxels from all the slices. Our implementation of brain extraction uses FSL's Brain Extraction Tool (BET) [10]. Among the four brain extraction utilities (SPM99 xbrain, BET, Brain Surface Extraction, McStrip), BET is the fastest algorithm available and it is also fairly accurate as well [4]. Most brain extraction tools like the SPM 99 `x_brain` generate brain mask from the anatomical scan and then apply the mask to the functional scans to strip away the skull and CSF voxels. However, BET can operate directly on functional scans which makes it ideal for online setup where only functional scans can enter the pipeline. The result of applying BET to a functional scan slice is shown in Figure 4.5.

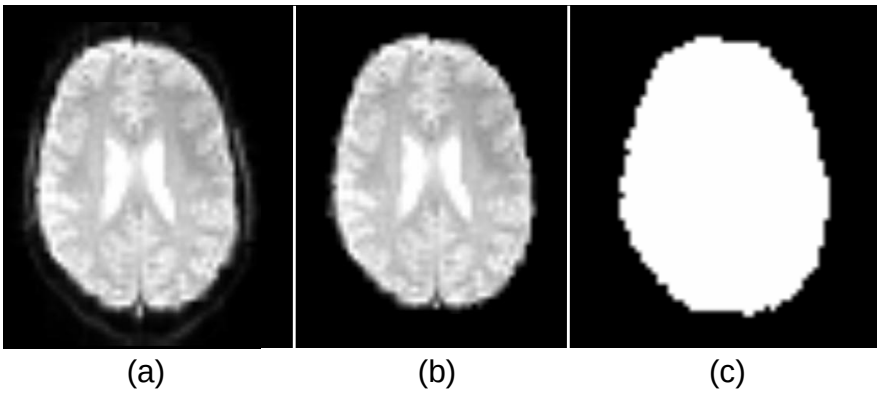

Figure 4.5 Brain extraction tool applied to a once slice of the functional scan (a): Shows the original slice before brain extraction (b) Shows the result after brain extraction (Fractional intensity threshold was set to 0.3) (c) shows the binary mask generated which is used to mask out non brain areas in all the subsequent scans

In our online fMRI setup, the very first functional scan in the FieldTrip buffer is fed to the BET algorithm which generates the binary mask, which is then applied to all subsequent scans to strip away non-brain voxels. The mask does not need to be recalculated in subsequent scans because if the motion correction is ON, then all the scans are aligned to the very first scan, which means that the mask for first scan is valid for all the rest of the scans as well. It is therefore important that if Brain Extraction is used, then motion correction should be used as well.

Brain extraction is just a one-time process that only occurs after the first scan arrives in the buffer. Therefore it adds no delay to the pipeline except for the approximately 2s that occurs during the first scan when the binary mask is being calculated. Since its only a one-time delay, the pipeline soon catches up.

Exactly how much skull is stripped depends on the *fractional intensity threshold* setting. Changing the fractional intensity threshold from its default value of 0.3 will cause the overall segmented brain to become larger (<0.3) or smaller (>0.3). This threshold must lie between 0 and 1. Currently, this parameter has been set to 0.3 because it generates the best results. This value is also recommended by the designers of BET tool for stripping skull from functional scans.

4.5.1 Dependencies

To be able to use Brain Extraction Tool, the BET utility should be present on the preprocessing system.

For Linux For Linux systems its important that full FSL should be installed.

For Windows For Windows systems its not important to install FSL. We will provide `bet.exe` files for both 32 and 64 bit systems. In case these files fail to run on your Windows platform, then you should probably install FSL[1].

Both Linux and Windows would also require the *NIFTI to Matlab* conversion tool which can be downloaded from *MATLAB CENTRAL* [2]. This tool is used because the BET utility only works with NIFTI files and hence we need to convert the pipeline data (which is MATLAB matrix format) to NIFTI format, apply BET to the NIFTI file generated, create the brain mask and then convert this brain mask from NIFTI format back to the Matlab data format.

4.6 ONLINE GLM FOR NUISANCE SIGNAL REMOVAL

General Linear Model (GLM) is a statistical model that explains data as a linear combination of explanatory variables, confounds and noise. The idea behind using GLM in the real-time pipeline is that we want to remove the contribution of known noise sources from a voxel time course, so that a voxel time course represent only the task related activity and unexplained noise. The explanatory variables are also called regressors and these need to be removed (or regressed out) from the voxel time courses. For this a GLM is used.

In an online GLM, the design matrix is built incrementally, scan by scan, as the data and regressors (such as motion estimates) become available. If the GLM has to be calculated at every scan, then after a few scans the design matrix would build up to such as extent that estimation of GLM model in real-time would not be possible using

[1]BET tool is also included in MRIcron and BioImage Suite distributions, so you can also use the BET utility from these software

classical approach. Therefore, a Recursive Least Squares (RLS) implementation [3] of GLM is applied to real-time fMRI data such that the duration of the experiment has no impact on the time required to calculate the beta estimates. This technique can reduce the memory requirements, because it is not necessary to store all the data as only the most recent scan data is used for updating the model. Our online GLM implementation offers three different configurations

```
cfg.numRegr = 1;  Regresses out constant offsets
cfg.numRegr = 2;  Regresses out constant offsets
                  and linear trends
cfg.numRegr = 5;  Regresses out constant offsets,
                  linear trends and translational motion
```

Recommended Settings We recommend to set `numReg` to either 2 or 5. Test your design with both 2 and 5 and then resort to the value which gives best possible results.

4.7 SPATIAL SMOOTHING

In spatial smoothing, neighboring voxel values are averaged which essentially low pass filters the data thereby removing high frequency fluctuations from the data. As a result sharpness of the image is reduced which is why this process is called smoothing. Smoothing is performed by convolving the fMRI signal with a Gaussian kernel (function) which has a specific width and shape of a normal distribution curve. The size of this smoothing kernel is specified by its Full Width Half Maximum measure. Full width at half maximum (FWHM) is an expression of the extent of a function, given by the difference between the two extreme values of the independent variable at which the dependent variable is equal to half of its maximum value.

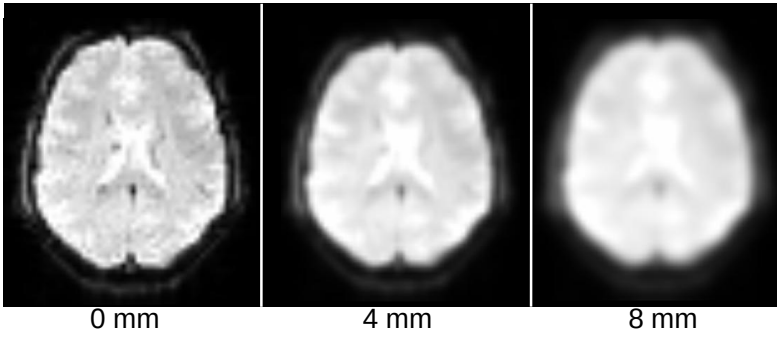

Figure 4.6 This figure demonstrates the effect of increasing the width of smoothing kernel on smoothness of functional data

Spatial smoothing is carried out on each volume of the fMRI data set separately. It's intended to reduce noise without reducing valid activation; this is successful as long as the underlying activation area is larger than the extent of the smoothing. Thus if you are looking for very small activation areas then you should maybe reduce smoothing to 5mm, and if you are looking for larger areas, you can increase it, maybe to 10 or even 15 mm. To turn off spatial smoothing simply set FWHM to 0. Figure 4.6 depicts the effect of increasing the smoothing from 0 mm to 8 mm. Any reduction in the random noise in the image will improve the ability of a statistical technique to detect true activations. Spatially smoothing each of the images improves the signal-to-noise ratio (SNR), but will reduce the resolution in each image, and so a balance must be found between improving the SNR and maintaining the resolution of the functional image.

Recommended Setting We have found that in real-time experiments, better results are obtained if no smoothing is applied. Therefore, we recommend users not to use the smoothing at all. To turn off smoothing set

```
cfg.smoothFWHM = 0;
```

CHAPTER 5

BCI MODULE

After the data has been processed through the preprocessing pipeline, its transferred to the BCI module. The BCI module is a separate computer on the network that has the following responsibilities:

- BCI loop control
- Feature extraction
- Classification and
- Dynamic adaptive stimulus generation

We will now discuss each of these functionalities in detail.

5.1 BCI LOOP CONTROL

The BCI module performs one of the most central role in the entire pipeline as it a controls the BCI Neurofeedback loop. To help facilitate in designing and writing

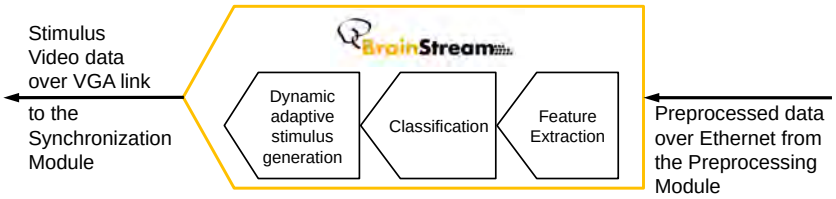

Figure 5.1 Block diagram of the Classification Module with it's various IO's

MATLAB based real-time experiments, we have made the BrainStream toolbox (`www.brainstream.nu`). BrainStream is a MATLAB-based software package for real-time processing of continuous data streams. In a real-time experiment, all processing steps and pipelines need to be executed at a specific time and in a specific order. This is handled by markers in BrainStream, which are received in synchrony with the data stream. Markers in themselves do not represent any kind of processing. Meaning is given to them by the actions (processing steps or pipelines) associated with the markers. Actions can be freely defined in order to accomplish some particular task, for example presenting stimuli to participants, numerical data processing, or controlling a data acquisition system. Incoming markers can initiate an unlimited sequence of actions at an unlimited number of time points. Users can control the insertion of markers and specify the actions associated with them.

With BrainStream almost any BCI experiment can be designed and controlled.

5.2 FEATURE EXTRACTION

Almost all BCI's require some kind of feature extraction. Features are individual measurable properties of the phenomena being observed. For fMRI, the easiest feature to extract is the voxel time course averaged over a few TRs. Other features such as t-statistics maps can be used as well [8]. BrainStream allows to write any MATLAB code for doing any type of feature extraction, hence the possibilities are endless.

5.3 CLASSIFICATION

Classification refers to an algorithmic procedure for assigning a given piece of input data into one of a given number of categories. BrainStream can call any MATLAB-based classification algorithms. In case the algorithms are written in C, then a MATLAB wrapper can be written to make it callable from BrainStream. There are

also some great MATLAB based toolboxes which can be used to classify neuroimaging data in real-time. One example is the *Donders Machine Learning Toolbox* [1] designed specifically to deal with high dimensional data such as fMRI. Because its a MATLAB-based toolbox it can be integrated with BrainStream code effortlessly.

5.4 DYNAMIC ADAPTIVE STIMULUS GENERATION

Stimulus generation is one of the most crucial component of a real-time BCI experiment. The neurofeedback in a real-time experiment needs to be constantly modified based on the subject performance, i.e. stimulus should be dynamically adapted based on the results of real-time data analysis. Using Psychtoolbox in conjunction with StimBox (a toolbox in progress using Psychtoolbox for developing dynamically updateable agenda-based stimulus presentation designs), any type of stimulus can dynamically generated. Again everything could be coded in MATLAB which makes using BrainStream such a great tool for real-time experiment control and execution.

Although BrainStream can be an invaluable tool in experiment control and execution, if required, the entire experiment can also be run without using BrainStream. It all depends on the prospective user's personal preference as to which approach is adapted. The pipeline itself is flexible enough to deal with almost any scenario.

CHAPTER 6

SYNCHRONIZATION MODULE

The synchronization module is Windows PC on the network which is connected by a serial COM port to the scanner. The scanner sends the trigger to the serial port every time a new scan is started. The TTL pulse received at the serial port is converted to FieldTrip event and sent to the preprocessing computer where it triggers the acquisition of a new volume into the raw data buffer.

Apart from that this module also receives a RESET pulse at its UDP port which resets the internal counter of the Fieldtrip event generation tool `serial_event.exe`

Furthermore, this is the only PC which is connected to the scanner projector. If the stimulus is not physically run on this computer then the VGA output from PC that generates the stimulus should be somehow routed to the projector. For this purpose, we use *Extron VGA mixer* which allow us to route either the VGA output from the Synchronization module or the BCI module to the scanner projector. If the stimulus is physically generated on the BCI module, then the VGA output from the BCI module is sent to the projector scanner. If the stimulus is generated on the Synchronization module then the VGA output of the synchronization module PC should be connected to the scanner.

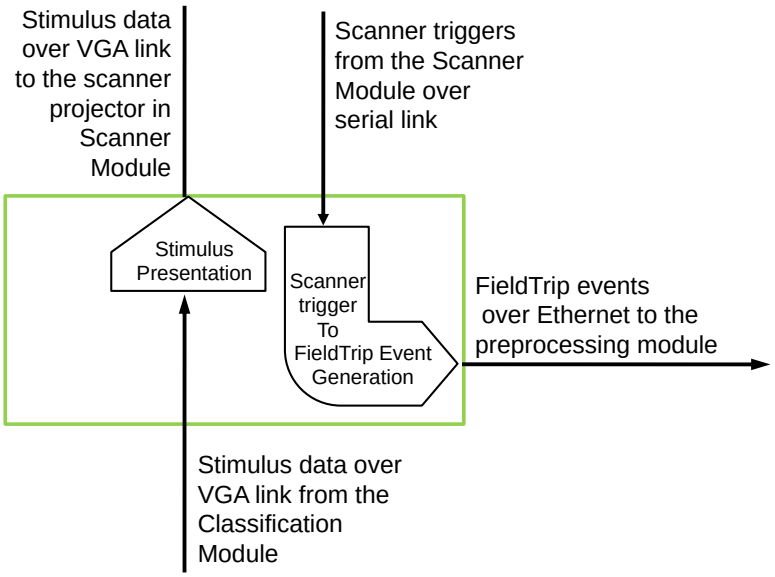

Figure 6.1 Block diagram of the Synchronization Module with it's various IO's

6.1 SERIAL EVENT TOOL

This section describes the tool serial_event, which is located in the directory `realtime/utilities/com2event`, and whose purpose is to write events to a FieldTrip buffer when a character is received on a serial port. The tool was developed specifically to in order to forward TTL pulses from the scanner to the FieldTrip buffer.

The user has the option to either only react on specific characters, and to write events with a fixed type and value, or to forward the received character as the value of the event. The sample field of the event can be auto-incremented, and reset by sending the string RESET to the UDP port (default = 1990) on which the tool listens. The latter feature is used in the DCCN's MRI lab to reset the sample counter when a new sequence is started on the scanner host (which is then picked up by the fMRI gui_streamer tool).

The tool is started from the command line by typing

serial_event [config-file]

If the config-file argument is not given, a default of serial_event.conf is assumed. Configuration file syntax

The operation mode of the tool is controlled by given a configuration file in ini-file syntax. A documented example follows below:

```
# Config file for Siemens 3t Scanner at Donders Institute
# Comment lines must start with a hash, empty lines are silently
# ignored

# buffer: FieldTrip buffer in the form hostname:port
# without quotes
buffer=lab-mri003:1972

# serial: Parameters of the serial port in the form
#   portname:baudrate:databits:stopbits:parity
# all numbers must be >=0, parity may only be 0 or 1
serial=COM7:115200:8:1:0

# character: specify single character to react on, or
# comment this out to react to every incoming character
character=a

# type: Type of event as either an integer, double
# precision number, or string (e.g. "serial")
type="serial"

# value: Value of event, can be integer/double/string
# or @ to pass on serial character
value="click"

# sample: number to transmit with first pulse plus
# increment per pulse, e.g. 0+1  (sends 0,1,2,3,...)
sample=-5+1

# offset and duration: integer numbers
duration=0
offset=0

# UDP port for RESET messages
port=1990
```

CHAPTER 7

REAL-TIME fMRI PROTOCOL: STEP-BY-STEP INSTRUCTIONS ON HOW TO START A REAL-TIME fMRI EXPERIMENT

Explained below is the procedure to start the real-time fMRI pipeline on the Siemens Trio 3T scanner at the DCCN. It is imperative that these steps be followed in the exact sequence as mentioned below. Failure to do so may result in unexpected outcomes.

7.1 STEP 1: PREPARING THE SCANNER

This step should be done on the scanner module.

- Register the subject in the scanner software. Then do all the necessary scanner routines.
- Perform a haste localizer scan which will help in correctly positioning the slices over the brain regions of interest. Refer to Figure 7.1 for information about how to correctly position slice matrix over brain matter.
- Do the anatomical scans, if necessary.

- Do the scanner shimming and make it fully ready for initiating a functional scan but DON'T yet start the functional scan.

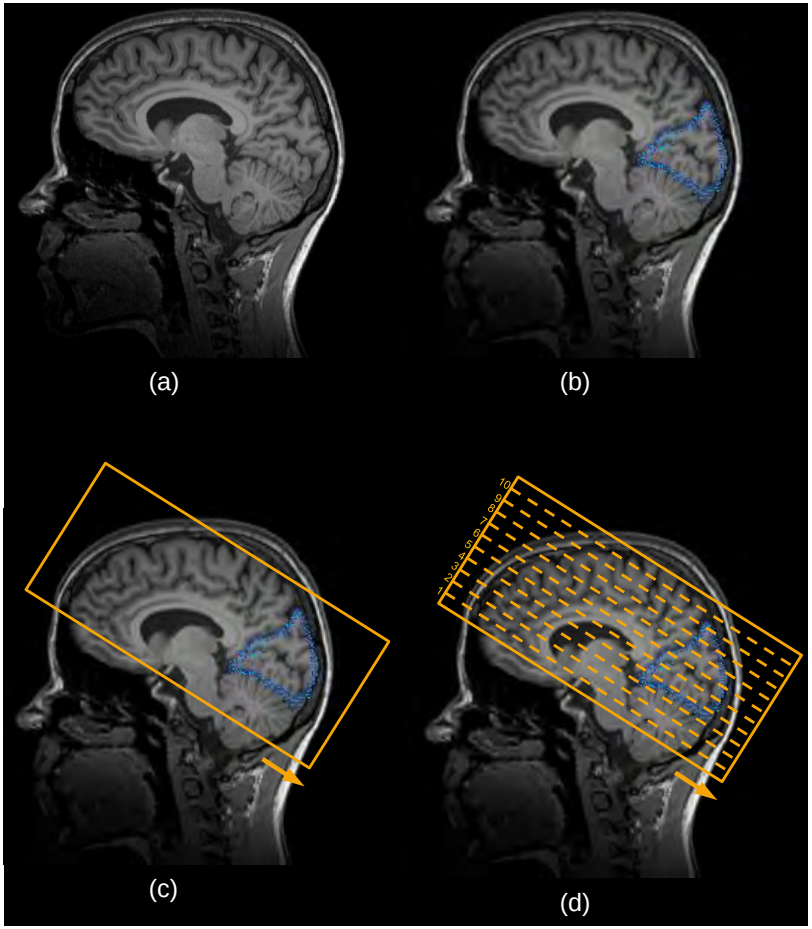

Figure 7.1 Correct slice positioning (a) Shows the brain image generated by the Haste localizer scan. The task is to correctly position the slices for functional data acquisition. The answer depends on the type of the experiment and the brain region of interest (b) Suppose it's a visual experiment involving the entire visual cortex which needs to be scanned. The visual cortex is shown in blue color (c) After we have located brain region of interest, position the slice matrix so that the region of interest is completely inside the slice matrix. If PACE is not used, then due to the head motion, the first and last few slices will get corrupt over time. It is therefore, imperative that the first three slices and last two slices do not cover the brain region of interest. (d) In this figure you can see that the first three slice (from bottom) and last two slice (top) do not cover the brain region of interest. So in case these slices do become corrupt due to subject motion, it would not hurt classification performance (provided these first three and last two slices are also masked in the preprocessing script, so that the voxels in these slices are completely removed from the training and test data)

7.2 STEP 2: STARTING THE BUFFERS

This step should be done on the preprocessing module.

1. Starting buffers on 'lab-mri003' computer
   lab_mri003 is a PC running Linux operating systems. Two FieldTrip buffers need to be spawned on this computer, each in its own separate terminal window. One buffer receives the Raw data stream from the scanner. The raw data is preprocessed. The raw and preprocessed data streams can be saved for later offline analysis.

   (a) Start raw data buffer

      i. Start the lab-mri003 computer. The computer will automatically login as *meduser*

      ii. Open terminal.

      iii. To store data you must Login to the storage servers using your DCCN Linux ID. Use the following command

      ```
      ssh adnnia@localhost
      ```

      Type in your credentials when asked for a username or password. For our case
      Username: adnnia
      Password: **********

      iv. `cd /home/common/matlab/fieldtrip/realtime/general`

      v. `./recording.glnxa64 ~adnnia/rawdata/ 1972`

      vi. If the folder already exist then an error will be displayed. In that case you should specify a new folder.

      vii. If later on something goes wrong, and the previously collected data needs to be deleted, then use this command

      ```
      rm -rf ~adnnia/rawdata/
      ```

      Warning: This will permanently delete your data so be careful while using this command.

   (b) Start preprocessed data buffer

      i. Open another terminal on the same computer

      ii. To store data you must Login again to the storage servers using your DCCN Linux ID. Use the following command

      ```
      ssh adnnia@localhost
      ```

Type in your credentials when asked for a username or password.
For our case
Username: adnnia
Password: **********

iii. `cd /home/common/matlab/fieldtrip/realtime/general`

iv. `./recording.glnxa64 ~adnnia/procdata/ 1973`

v. If later on something goes wrong, and the previously collected data needs to be deleted, then use this command

`rm -rf ~adnnia/procdata/`

Warning: This will permanently delete your data so be careful while using this command.

7.3 STEP 3: STARTING THE SERIAL EVENT TOOL

This step should be performed on the synchronization module.

Start pulse tool on Presentation machine 'Presentation009'

(a) On *Presentation009* machine, go to the following folder

`D:\TTL_to_FieldTrip`

(b) Open config file to check settings (of serial_event tool).

(c) Sample = 0 + 1 (could be -5 + 1 to drop the first few dummy scans. However, it's better to not use those scans in the realignment of the preprocessing pipeline. The latter is not implemented yet).

(d) Port should be 1990. This number should be the same in the GUI_streamer.

(e) Click on .exe file (of serial_event tool)

7.4 STEP 4: STARTING THE PREPROCESSING PIPELINE

This step should be done on the preprocessing module.

Start Matlab session on 'lab_mri003' computer

(a) Open terminal

(b) Type matlab &

(c) `cd /home/common/matlab/fieldtrip/realtime/online_mri`

(d) Type

```
cfg = [];
```

(e) Type type ft_omri_pipeline.m to check config settings

(f) Type ft_omri_pipeline(cfg)

(g) Press enter

7.5 STEP 5: MAKING THE REAL-TIME EXPORT CONNECTION

This step should be done on the scanner module.

Start GUI_streamer on scanner host computer

(a) Press CTRL+ Esc

(b) Click on shortcut to GUI_streamer.exe

(c) Port should be 1972 (FieldTrip raw buffer port) and the target should be lab-mri003

(d) UDP target hostname = Presentation009

(e) UDP port should be 1990

(f) When connections are enabled, they turn green!

(g) If at any point the connection gets disconnected, press 'CONNECT' again to reconnect.

7.6 STEP 6: STARTING THE fMRI SEQUENCE

This step should be done on the scanner module. With the help of the scanner software send a 'RESET' pulse. Your MRI technician will probably know how to do it. Once the 'RESET' pulse is sent, it should be visible GUI of the serial_event tool on Presentation009 machine. After that, the fMRI sequence should be started. The output of the serial_event tool should look the same as shown in Figure 7.2.

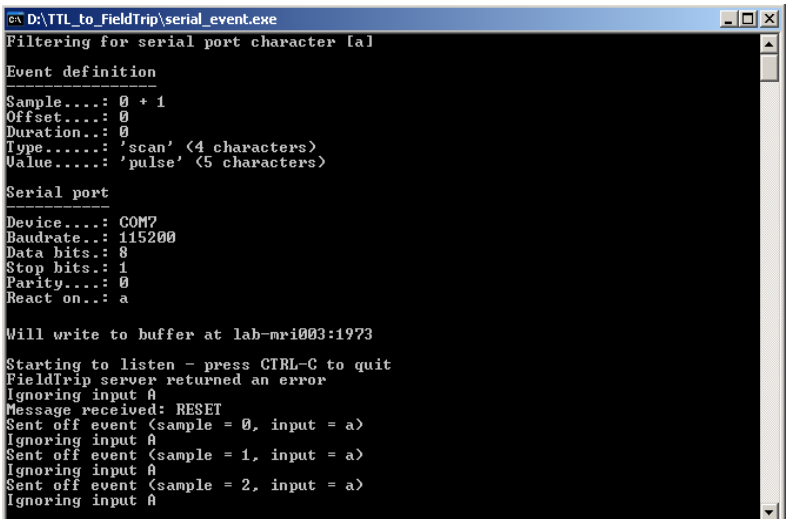

Figure 7.2 A view of the serial event after the fMRI sequence has been initiated. The sequence is started by first sending a RESET pulse. Normal scanner pulse should come afterwards this RESET message

7.7 STEP 7: STARTING THE BCI EXPERIMENT

This step should be done on the BCI module. It involves starting BrainStream and the experiment files.

7.8 STEP 8: ENDING THE EXPERIMENT

- Close BrainStream
- Close both buffers by pressing CTRL+C.
- Close the serial event tool
- Close the MATLAB session on the preprocessing PC

CHAPTER 8

EVALUATION

To evaluate the performance of the real-time pipeline we ran it on a Laptop with following specification:

- Intel(R) Core(TM)2Duo CPU T6600 @2.20GHz
- 4GB RAM
- Windows 7 64-bit operating system
- MATLAB R2012a

The following results were obtained by running the pipeline

- Time taken by brain extraction = 2s
  This is a one time operation and is performed only in the beginning of the experiment. The pipeline is able to catch up within the next 5TRs.
- Time taken by slice time correction (Linear interpolation) = 0.1s
- Time taken by motion correction = 0.28s

- Time taken by online GLM (5 regressors) = 0.25s
- Time taken by spatial smoothing (8mm smoothing kernel) = 0.1s

Total pipeline processing delay = Time taken by slice time correction + Time taken by motion correction + Time taken by online GLM + Time taken by spatial smoothing

Total pipeline processing delay = 0.1s + 0.28s + 0.25s +0.1s = 0.73s

8.1 CONCLUSION

The real-time fMRI pipeline developed is able to process each scan within a 1s time window. This means that a TR as low as 1 second can be used in the real-time fMRI experiments. Our evaluation was carried on a low end laptop computer. This shows that our pipeline does not require any significant computational resources to do its job and will work on any reasonably decent workstation.

CHAPTER 9

FUTURE IMPROVEMENTS

Although the pipeline is fully functional, there is much to be desired as to the ease of using it. Here are the few suggestions for future improvements:

- The main workhorse of the pipeline (`ft_omri_pipeline_nuiscane.m`) should be incorporated in a GUI where all the configuration settings could be conveniently made before the start of the experiment. This will save a lot time which is currently wasted in just make theses settings in `cfg` structure. Furthermore, it will make the pipeline more attractive for the prospective users.
- Sinc interpolation for slice time correction should be further investigated to determine its advantage over the linear interpolation. Furthermore, the Sinc interpolation code needs optimization so as to reduce its execution time to below 1s.
- Online GLM implementation should be improved such that regressors could be added dynamically during the experiment execution. Currently, the number of regressors are specified before the start of the experiment but they cannot be changed during the experiment. Adding this feature will bring the pipeline on par with Turbo BrainVoyager.

- The online GLM should allow the possibility to select region of interests and to confine the analysis to these region of interests.
- A GUI should also be designed for starting the two buffers on the preprocessing PC. Currently starting these two buffers takes too much typing time before the start of the experiment. A convenient GUI where each user can save and retrieve there path setting, would save so much time.
- There should be an option to see the output of all the different stages of the pipeline. Currently, the user can only see the final preprocessed scans. It would be great if the output of each stage of the pipeline could be displayed as per the user's choice.

REFERENCES

1. Donders Machine Learning Toolbox. `http://sourceforge.net/projects/dmlt/`.
2. Tools for NIfTI and ANALYZE image. `http://www.mathworks.com/matlabcentral/fileexchange/8797`.
3. E Bagarinao, K Matsuo, T Nakai, and S Sato. Estimation of general linear model coefficients for real-time application. *NeuroImage*, 19(2):422–429, 2003.
4. Kristi Boesen, Kelly Rehm, Kirt Schaper, Sarah Stoltzner, Roger Woods, Eileen Lüders, and David Rottenberg. Quantitative comparison of four brain extraction algorithms. *NeuroImage*, 22(3):1255–1261, 2004.
5. R Decharms, K Christoff, G Glover, J Pauly, S Whitfield, and J Gabrieli. Learned regulation of spatially localized brain activation using real-time fMRI. *NeuroImage*, 21(1):436–443, 2004.
6. R Christopher deCharms, Fumiko Maeda, Gary H Glover, David Ludlow, John M Pauly, Deepak Soneji, John D E Gabrieli, and Sean C Mackey. Control over brain activation and pain learned by using real-time functional MRI. *Proceedings of the National Academy of Sciences of the United States of America*, 102(51):18626–18631, 2005.
7. G C Feigl, S Safavi-Abbasi, A Gharabaghi, V Gonzalez-Felipe, A El Shawarby, H-J Freund, and M Samii. Real-time 3T fMRI data of brain tumour patients for intra-operative localization of primary motor areas. *European journal of surgical oncology the journal of the European Society of Surgical Oncology and the British Association of Surgical Oncology*, 34(6):708–715, 2008.
8. Bo Jin, Alvin Strasburger, Steven J Laken, F Andrew Kozel, Kevin A Johnson, Mark S George, and Xinghua Lu. Feature selection for fMRI-based deception detection. *BMC Bioinformatics*, 10(Suppl 9):S15, 2009.
9. Chandrasekharan Kesavadas, Bejoy Thomas, Sreedharan Sujesh, Radhakrishnan Ashalata, Mathew Abraham, Arun Kumar Gupta, and Kurupath Radhakrishnan. Real-time functional MR imaging (fMRI) for presurgical evaluation of paediatric epilepsy. *Pediatric Radiology*, 37(10):964–974, 2007.
10. Stephen M Smith. Fast robust automated brain extraction. *Human Brain Mapping*, 17(3):143–155, 2002.
11. Larry Stanford. Applications of real-time fMRI : Pain treatment and substance abuse treatment. *Work*, (March), 2007.
12. S Thesen, O Heid, E Mueller, and L R Schad. Prospective acquisition correction for head motion with image-based tracking for real-time fMRI. *Magnetic Resonance in Medicine*, 44(3):457–465, 2000.
13. Nikolaus Weiskopf, Frank Scharnowski, Ralf Veit, Rainer Goebel, Niels Birbaumer, and Klaus Mathiak. Self-regulation of local brain activity using real-time functional magnetic resonance imaging (fMRI). *Journal of Physiology Paris*, 98(4-6):357–373, 2004.
14. Nikolaus Weiskopf, Ranganatha Sitaram, Oliver Josephs, Ralf Veit, Frank Scharnowski, Rainer Goebel, Niels Birbaumer, Ralf Deichmann, and Klaus Mathiak. Real-time functional magnetic resonance imaging: methods and applications. *Magnetic Resonance Imaging*, 25(6):989–1003, 2007.