MRI Together Workshop 2021

Das Deutsche Zentrum für
Neurodegenerative Erkrankungen

Raw Data Conversion

December 14 (UTC), 2021

Philipp Ehses

Speaker name:
Philipp Ehses

Conflicts of interest regarding this presentation:
Nothing to disclose

ESMRMB
European Society for Magnetic Resonance in Medicine and Biology

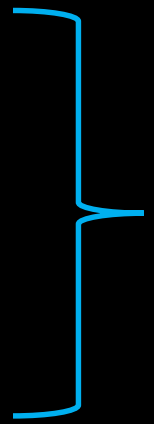
Overview

- Why should we care about MRI raw data?
- Why should we aim for open standards?
- The ISMRM raw data format (MRD)
  - File structure and recon. example
- Self-promotion: twixtools python reader
  - Open tool to read (and write) a non-open format (Siemens raw data)
  - Entry point for open-source reco. pipeline

    (includes conversion utility to bart)

Terminology: What is MRI raw data?

- Often means “raw images”
  - i.e. pre-analysis images (e.g. in fMRI, DTI, ...)
  - This is **NOT** what this talk is about!
- This talk is about **pre-reconstruction data**
  - “k-space data”
- Raw data consists of
  - Protocol information (strings)
    - patient name, sequence parameters, etc.
  - Acquired data from ADC (complex floats)
    - usually with metadata
      (line counters, slice position, time stamps, ...)



Stored in one or multiple files

Why should we care about MRI raw data?

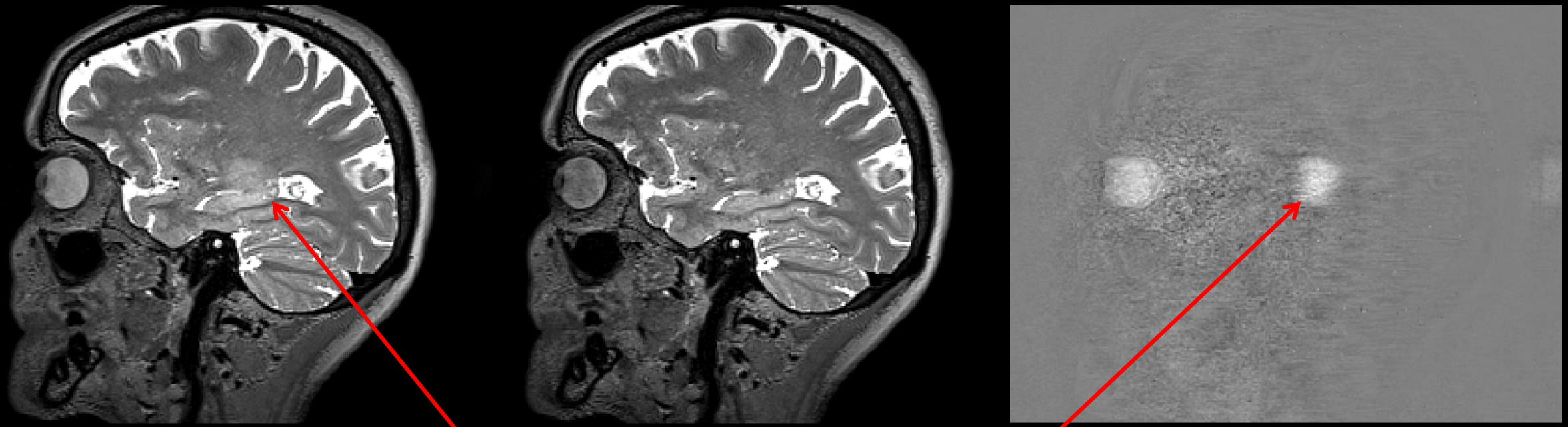
- Benefit from latest reconstruction algorithms / develop your own
  - Long-term storage: Even old data can benefit from improvements!
    (Example: Human Connectome project)
- Look for sources of image artifacts
  - Often easier in raw data + retrospective correction possible (Example: next slide)
  - QA (Quality assurance) protocols
- Calibrate something
  - Trajectory / gradient delays
  - Gradient-Impulse-Response Function (GIRF)
  - Field probe system
  - …

Rhineland Study*: Artifact in T_2-weighted SPACE

Original SPACE

Retrospective Reconstruction
(Exclude Grappa autocalibration data)

Difference



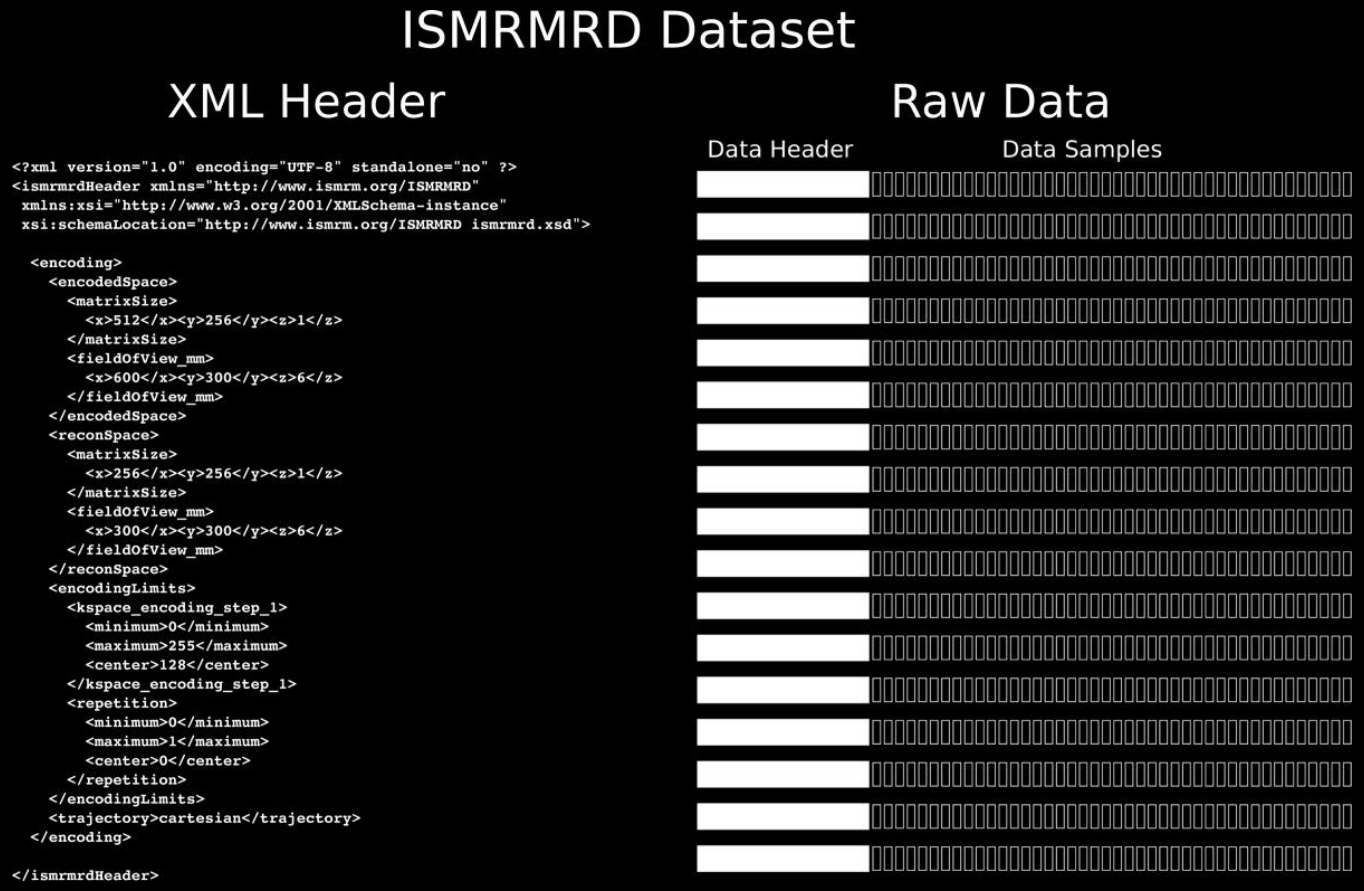
Eye movement causes fold-over artifact
(parallel imaging)

*The Rhineland Study (start in 2016) is a prospective cohort study with up to 30,000 participants that assesses physical and mental health over decades. A 1h MRI exam is included in the study protocol.

Why should we aim for open standards?

- Easy to share data
  - Open image standards already common (DICOM, nifti, ...)
- Easy to share code and reconstruction pipelines
  - Less IP right issues
  - Lower barriers between systems from different manufacturers
- Potential to increase Reproducibility
  - between sites
  - between vendors

The ISMRM Raw Data format (MRD/ISMRMRD)



- Initiated by NIH
- API: https://ismrmrd.github.io/
- Code: https://github.com/ismrmrd
  - C++, Python, MATLAB Code & Examples
  - Raw Data Converters for Bruker, GE, Philips, Siemens
- Growing support
  - Gadgetron
  - MRIReco.jl
  - BART (preliminary)

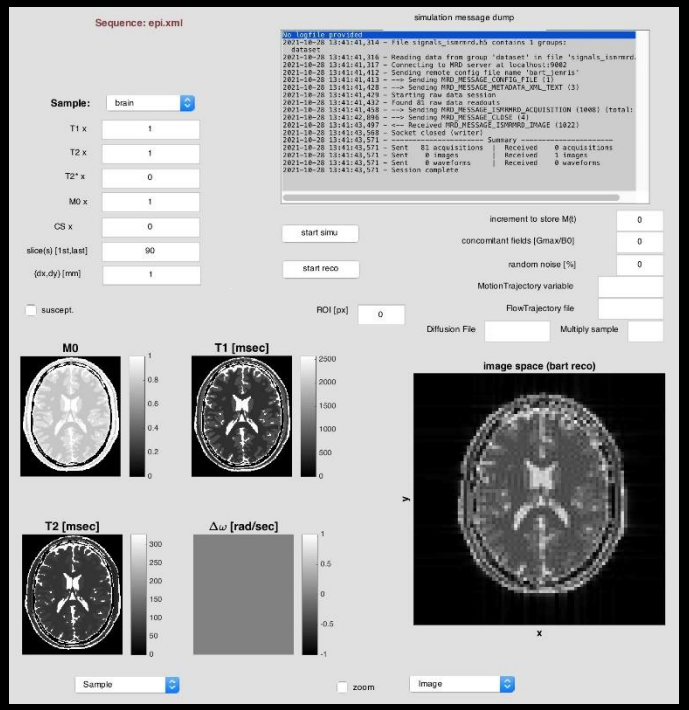
Ahsan Javed and Raj Ramawawmy:

“Hands-on with Vendor-agnostic MRI data: conversion to MRD”

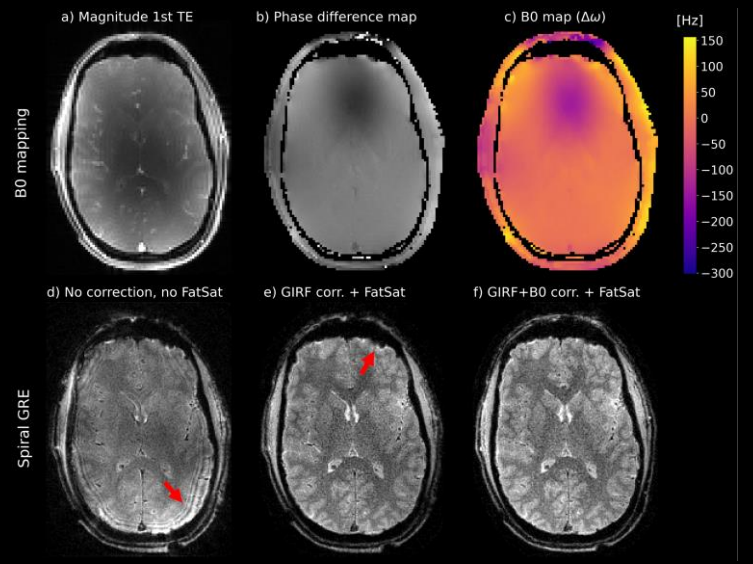
Later in this session

Inati et al. “ISMRM Raw data format”. MRM 2016.

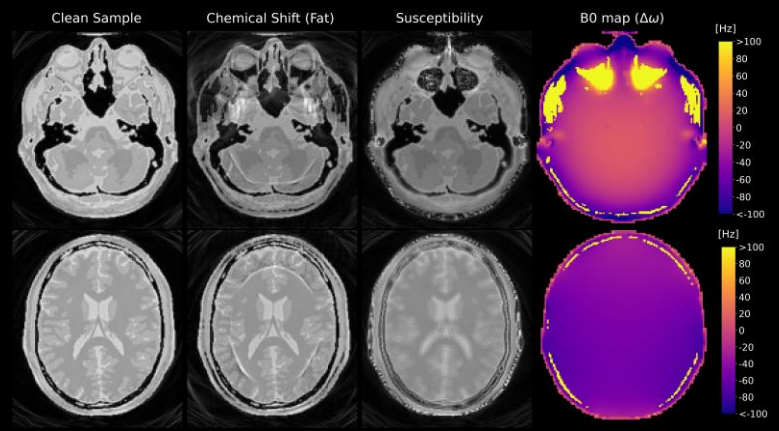
JEMRIS now supports MRD export & recon.



JEMRIS GUI



PulSeq[1] Acquisition



JEMRIS[2] simulation with same MRD-based reconstruction pipeline

Marten Veldmann:

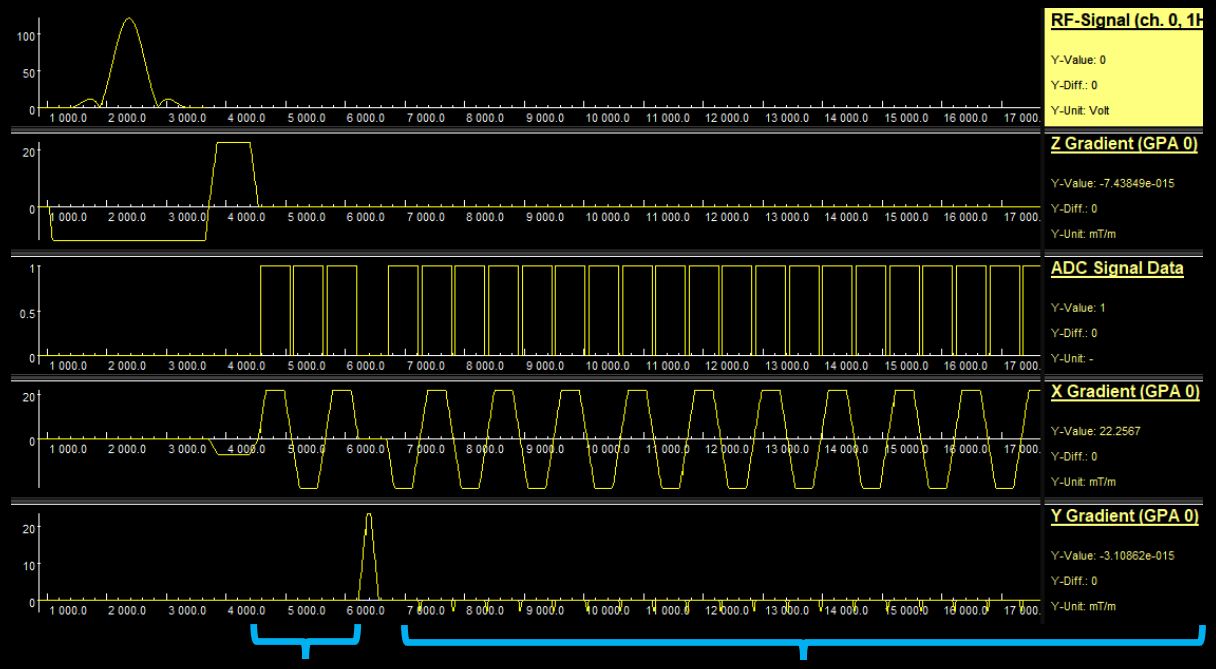
“Integrated Acquisition/Simulation/Reconstruction”

Wednesday Dec. 15, Indian: 6:00-8:00 UTC

[1]https://pulseq.github.io/
[2]https://www.jemris.org/

MRD file example: EPI

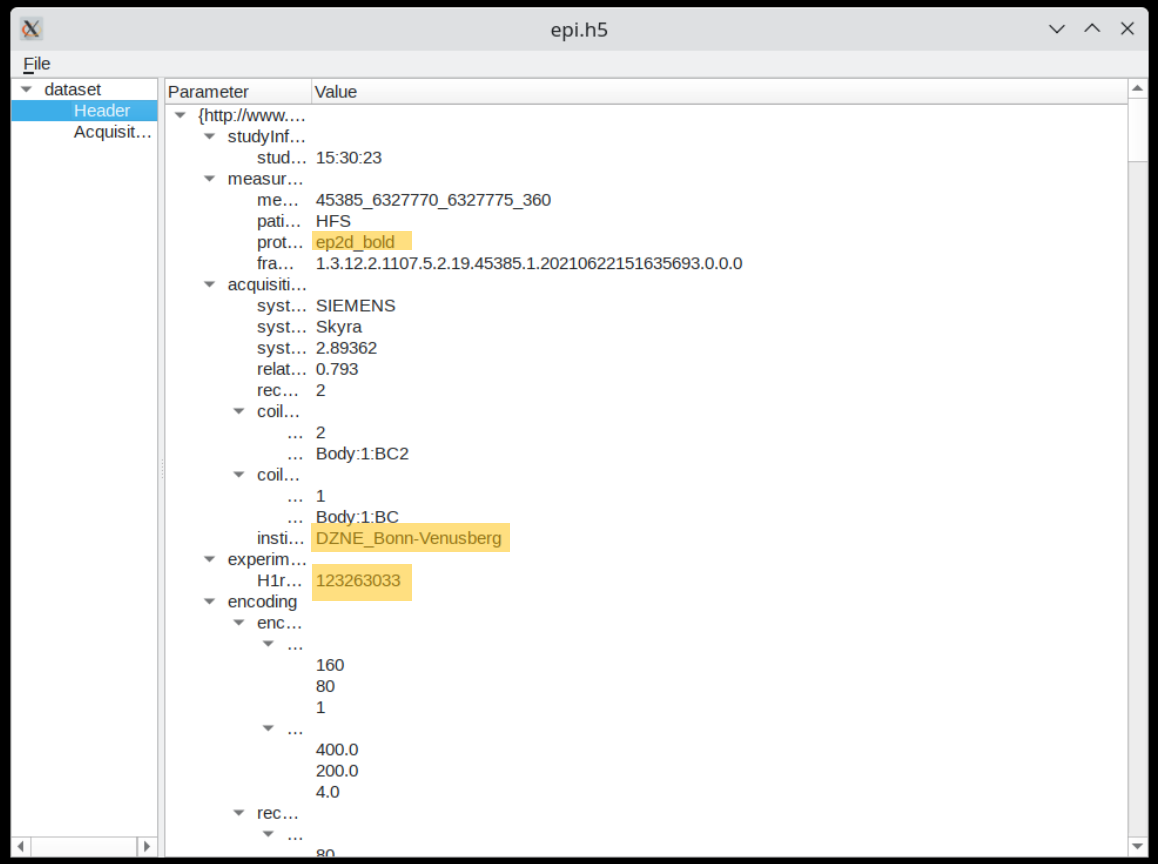
Sequence Diagram



phase-correction scans

single-shot EPI readout

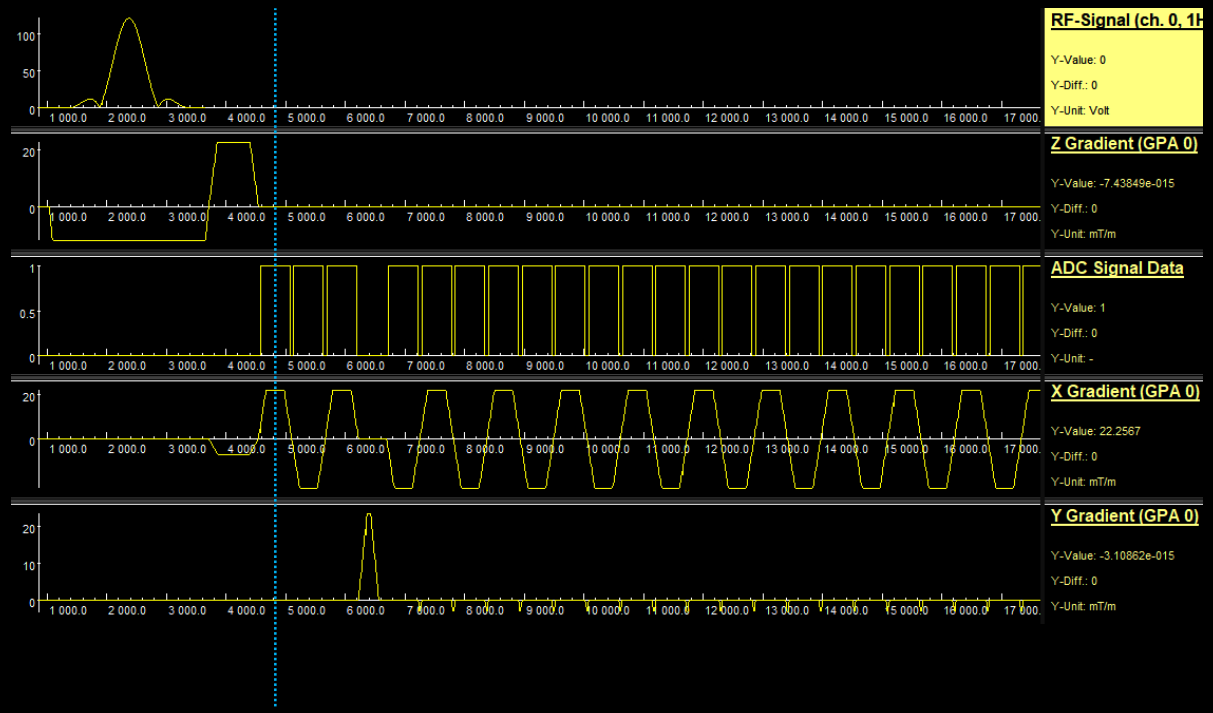
ISMRMRD file viewer



https://github.com/ismrmrd/ismrmrdviewer

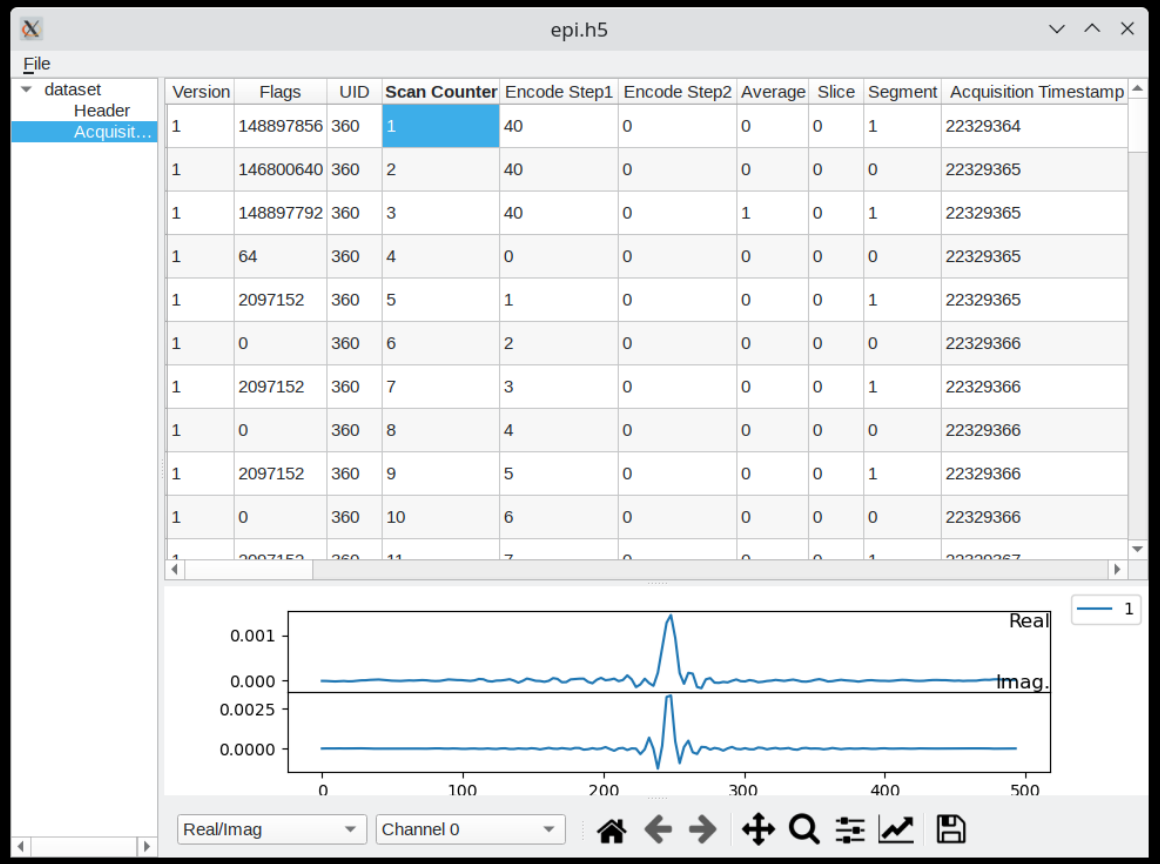
MRD file example: EPI

Sequence Diagram



position in file

ISMRMRD file viewer

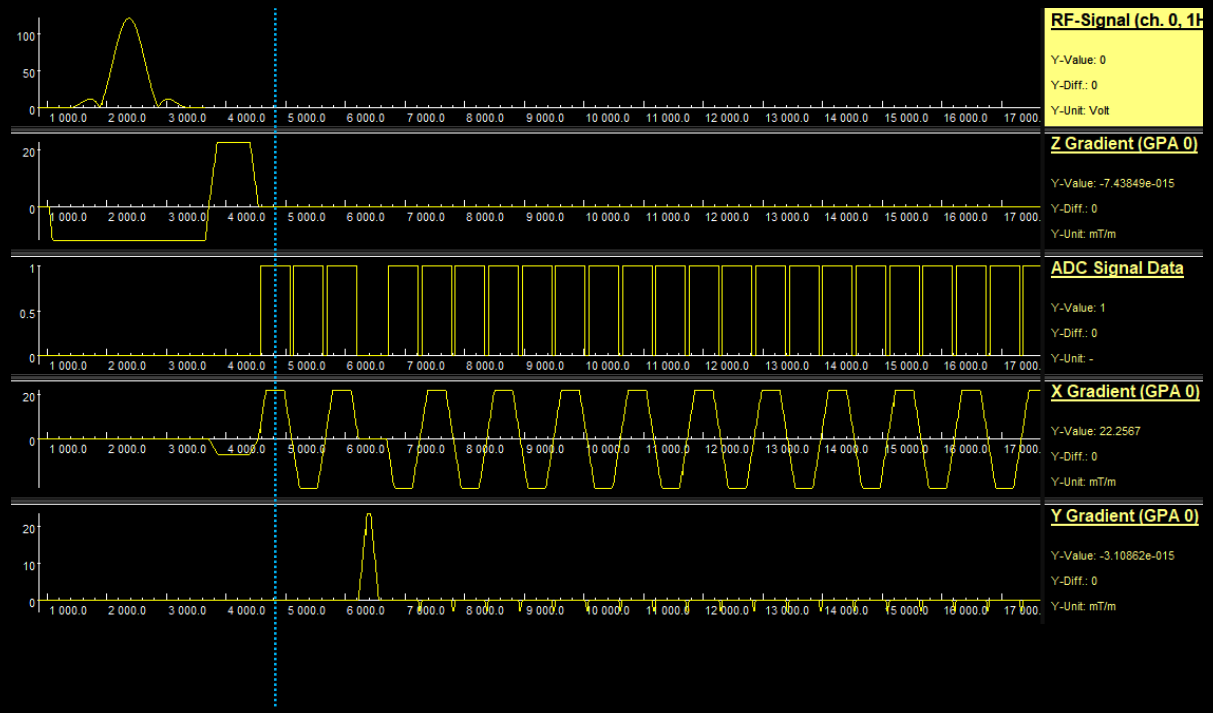


Version	Flags	UID	Scan Counter	Encode Step1	Encode Step2	Average	Slice	Segment	Acquisition Timestamp
1	148897856	360	1	40	0	0	0	1	22329364
1	146800640	360	2	40	0	0	0	0	22329365
1	148897792	360	3	40	0	1	0	1	22329365
1	64	360	4	0	0	0	0	0	22329365
1	2097152	360	5	1	0	0	0	1	22329365
1	0	360	6	2	0	0	0	0	22329366
1	2097152	360	7	3	0	0	0	1	22329366
1	0	360	8	4	0	0	0	0	22329366
1	2097152	360	9	5	0	0	0	1	22329366
1	0	360	10	6	0	0	0	0	22329366

https://github.com/ismrmrd/ismrmrdviewer

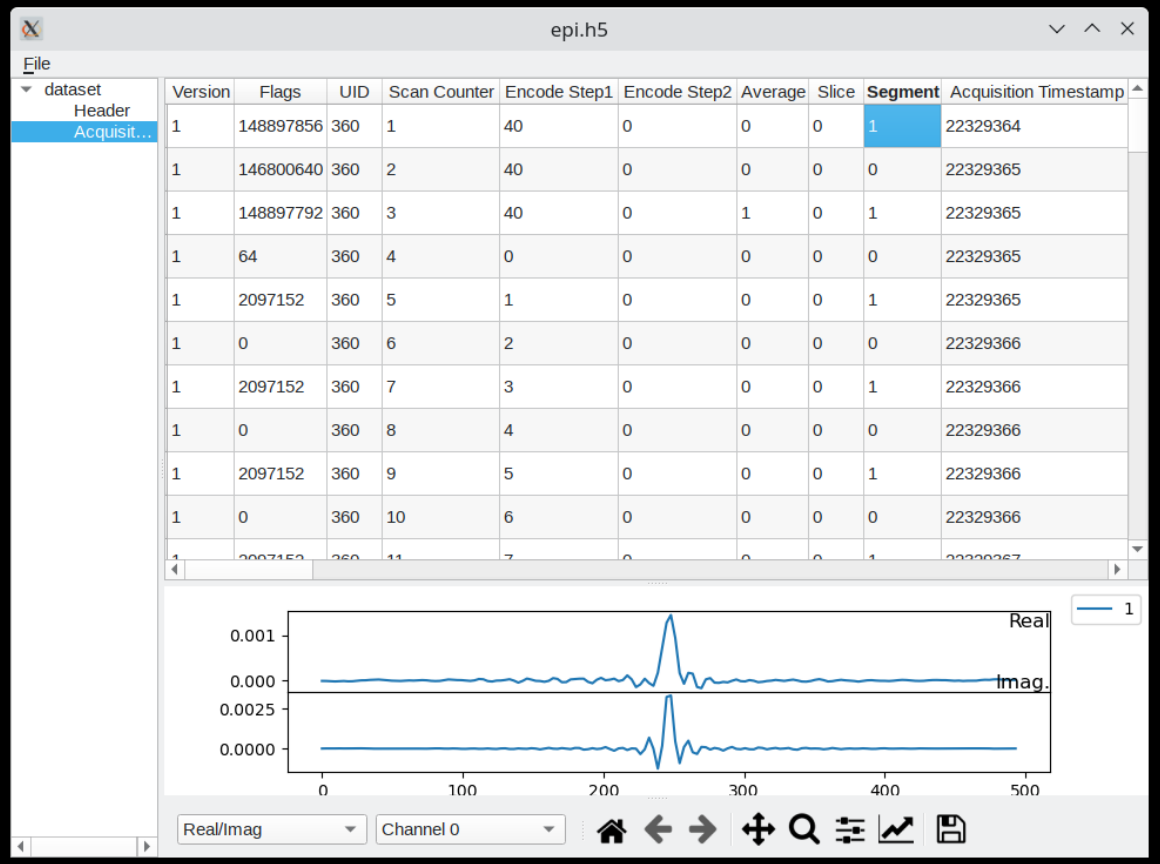
MRD file example: EPI

Sequence Diagram



position in file

ISMRMRD file viewer

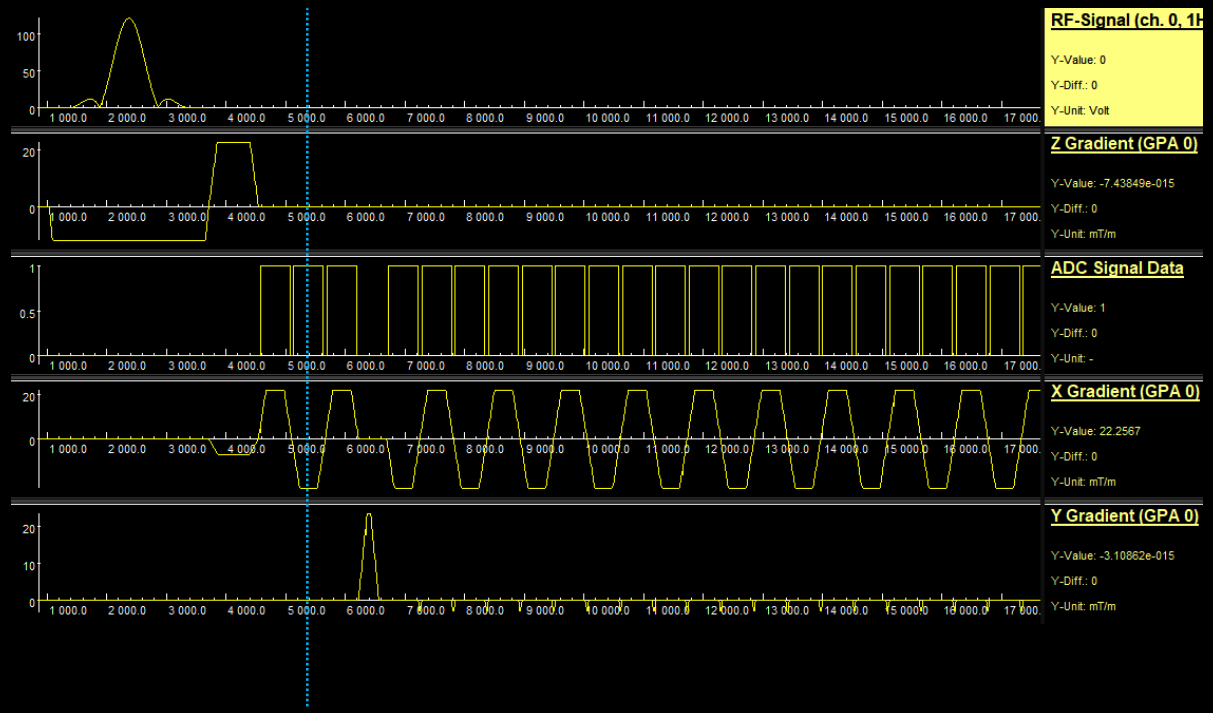


Version	Flags	UID	Scan Counter	Encode Step1	Encode Step2	Average	Slice	Segment	Acquisition Timestamp
1	148897856	360	1	40	0	0	0	1	22329364
1	146800640	360	2	40	0	0	0	0	22329365
1	148897792	360	3	40	0	1	0	1	22329365
1	64	360	4	0	0	0	0	0	22329365
1	2097152	360	5	1	0	0	0	1	22329365
1	0	360	6	2	0	0	0	0	22329366
1	2097152	360	7	3	0	0	0	1	22329366
1	0	360	8	4	0	0	0	0	22329366
1	2097152	360	9	5	0	0	0	1	22329366
1	0	360	10	6	0	0	0	0	22329366

https://github.com/ismrmrd/ismrmrdviewer

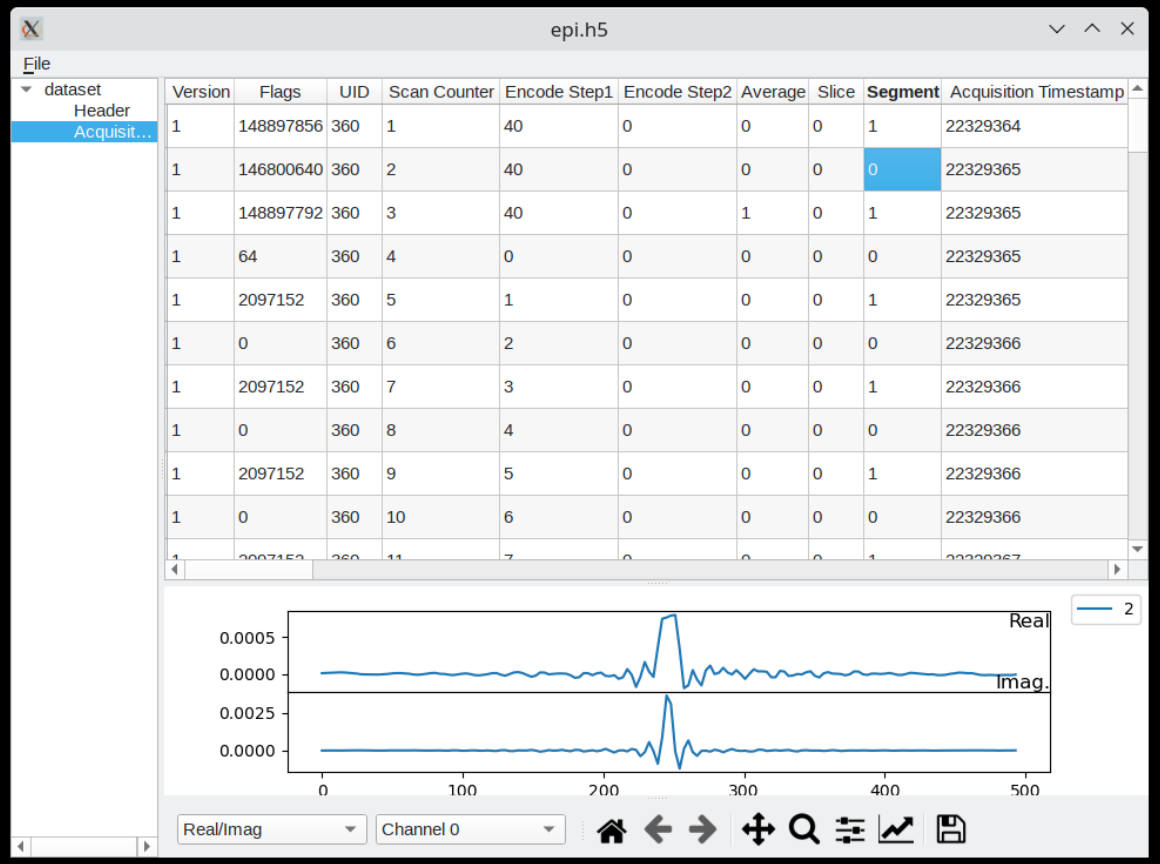
MRD file example: EPI

Sequence Diagram



position in file

ISMRMRD file viewer

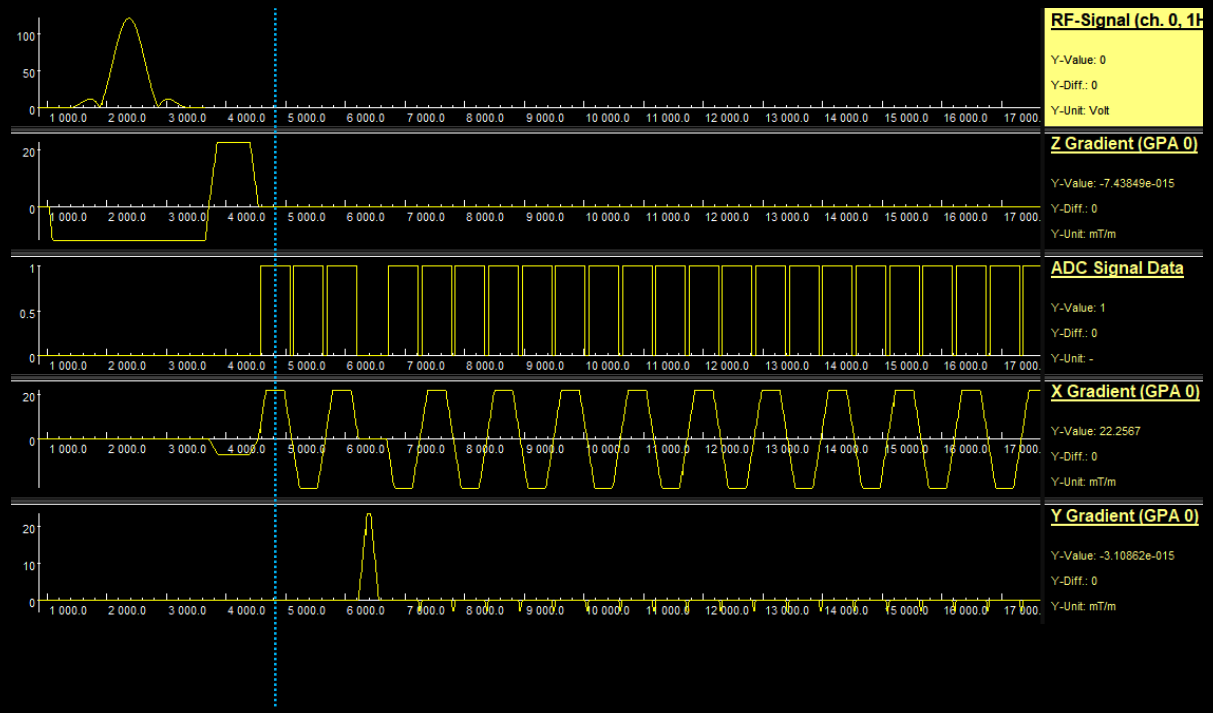


Version	Flags	UID	Scan Counter	Encode Step1	Encode Step2	Average	Slice	Segment	Acquisition Timestamp
1	148897856	360	1	40	0	0	0	1	22329364
1	146800640	360	2	40	0	0	0	0	22329365
1	148897792	360	3	40	0	1	0	1	22329365
1	64	360	4	0	0	0	0	0	22329365
1	2097152	360	5	1	0	0	0	1	22329365
1	0	360	6	2	0	0	0	0	22329366
1	2097152	360	7	3	0	0	0	1	22329366
1	0	360	8	4	0	0	0	0	22329366
1	2097152	360	9	5	0	0	0	1	22329366
1	0	360	10	6	0	0	0	0	22329366

https://github.com/ismrmrd/ismrmrdviewer

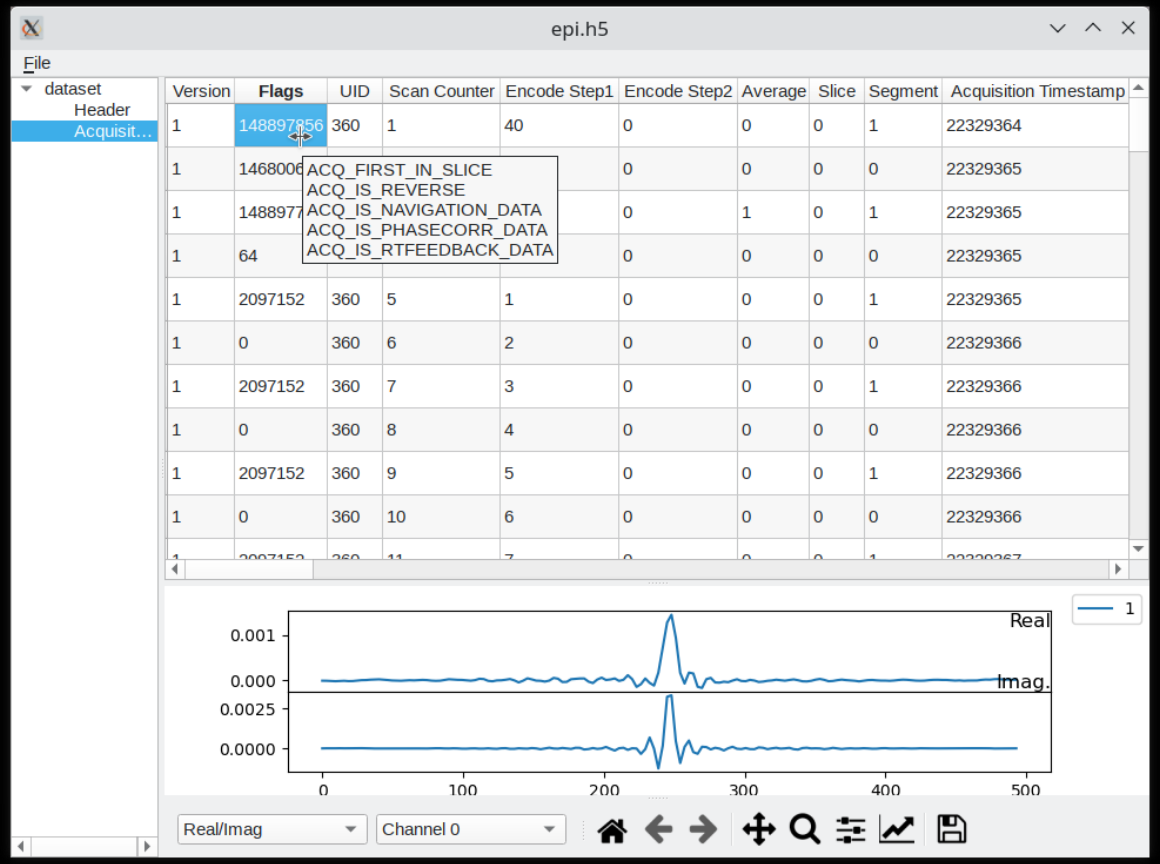
MRD file example: EPI

Sequence Diagram



position in file

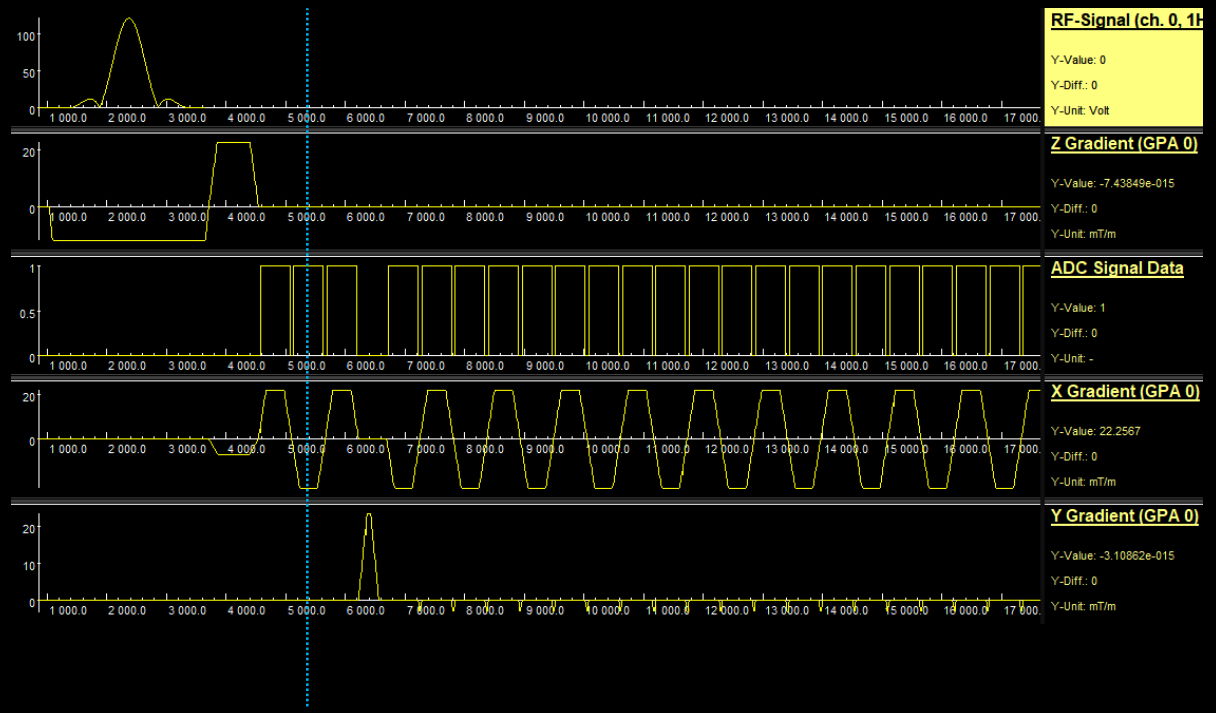
ISMRMRD file viewer



https://github.com/ismrmrd/ismrmrdviewer

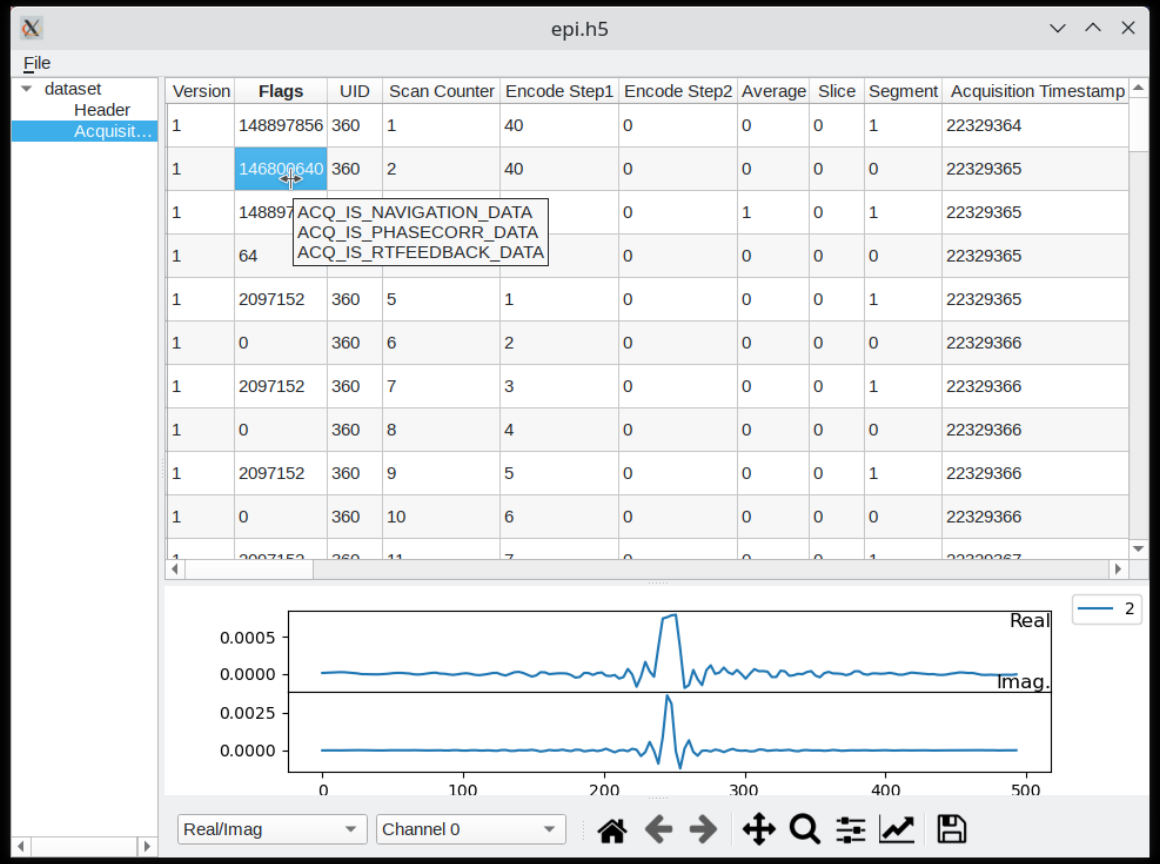
MRD file example: EPI

Sequence Diagram



position in file

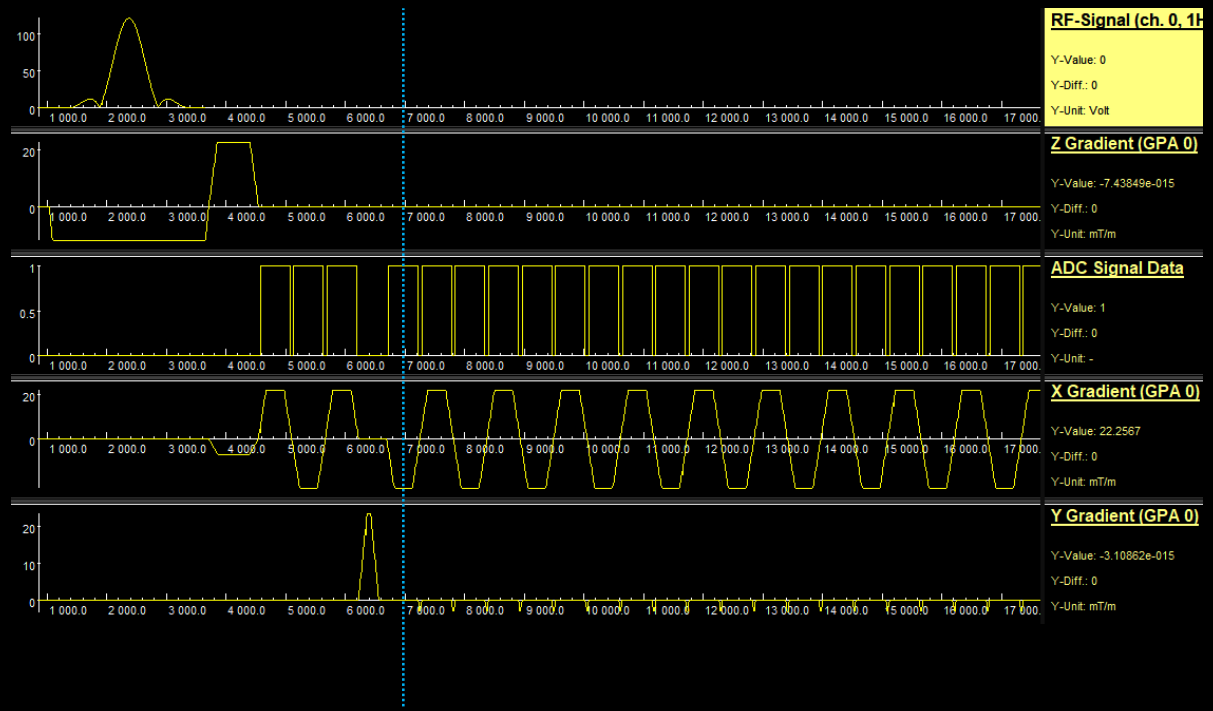
ISMRMRD file viewer



https://github.com/ismrmrd/ismrmrdviewer

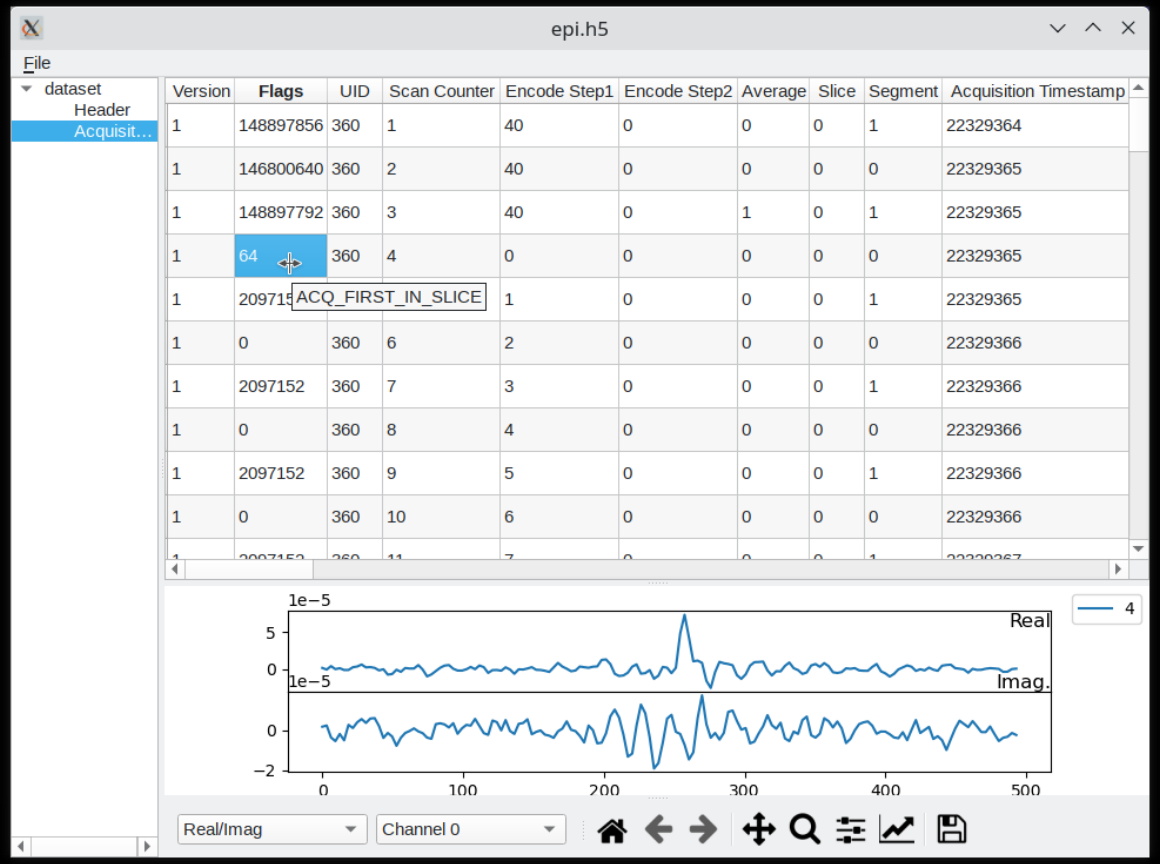
MRD file example: EPI

Sequence Diagram



position in file

ISMRMRD file viewer



https://github.com/ismrmrd/ismrmrdviewer

MRD python reconstruction example

```python
import ismrmrd
import numpy as np

# helper functions
def ifftnd(kspace, ax=[-1]):
    return np.fft.fftshift(np.fft.ifftn(np.fft.ifftshift(kspace, axes=ax), axes=ax), axes=ax)
def rms_combination(sig, axis=1):
    return np.sqrt(np.sum(abs(sig)**2, axis))

# parse the raw data file
dset = ismrmrd.Dataset('gre.h5', 'dataset', create_if_needed=False)

# determine k-space shape and sort all 'imaging' acq. into a k-space array
image_acqs = [dset.read_acquisition(key) for key in range(dset.number_of_acquisitions())]
n_channel, n_column = image_acqs[0].data.shape
n_line = 1 + max([acq.idx.kspace_encode_step_1 for acq in image_acqs])
kspace = np.zeros([n_line, n_channel, n_column], dtype=np.complex64)
for acq in image_acqs:
    kspace[acq.idx.kspace_encode_step_1] = acq.data

image = rms_combination(ifftnd(kspace, [0,-1]))  # reconstruct image

print('k-space shape: %d lines x %d coils x %d columns'%(kspace.shape))
```

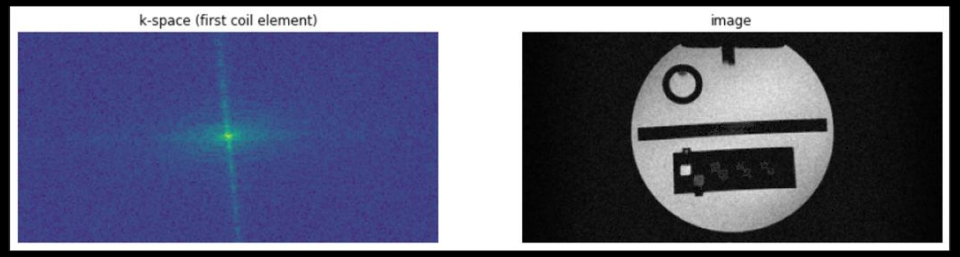
Imports and helper functions

Open the file

read & sort

reconstruct

```
k-space shape: 160 lines x 2 coils x 320 columns
```



- python interface provides access to protocol and acquisition blocks
- Data needs to be sorted into k-space array based on loop counters (and possibly additional flags)
- Most data include 2x oversampling (from a Siemens perspective)

https://github.com/ismrmrd/ismrmrd-python
https://github.com/ismrmrd/ismrmrd-python-tools

twixtools

Python redesign of the matlab Siemens reader “mapVBVD”

Modular design:

- read_twix: low(er)-level access to acquisition blocks (“mdb”s)
  - Interface very similar to ismrmrd-python
  - Basis for higher-level tools
- write_twix:
  - Limited support for writing modified data back to Siemens raw data format
  - Modified file can be reconstructed on scanner (retro recon)
- map_twix: sorts data into k-space arrays
  - returns memory-mapped array objects with numpy-like slicing support
  - separate arrays for different data types (image, phase corr., noise, ...) based on acq. flags
  - supports simple pre-processing steps: zero-filling to target resolution, os-removal, ...
- Open for suggestions & pull requests (see github address below)

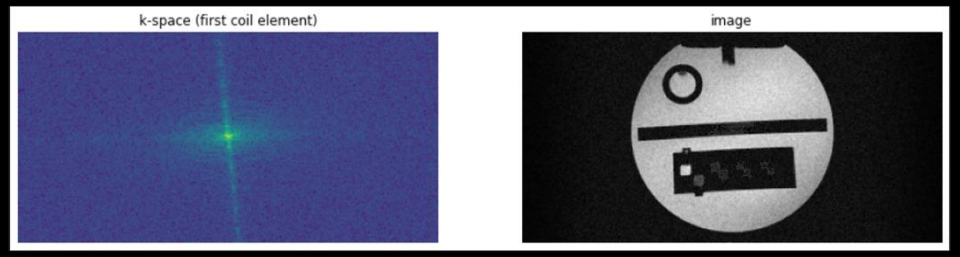
```
pip install git+https://github.com/pehses/twixtools.git
```

ISMRMRD vs. twixtools example

ismrmrd-python

```python
import ismrmrd
import numpy as np

# helper functions
def ifftnd(kspace, ax=[-1]):
    return np.fft.fftshift(np.fft.ifftn(np.fft.ifftshift(kspace, axes=ax), axes=ax), axes=ax)
def rms_combination(sig, axis=1):
    return np.sqrt(np.sum(abs(sig)**2, axis))

# parse the raw data file
dset = ismrmrd.Dataset('gre.h5', 'dataset', create_if_needed=False)

# determine k-space shape and sort all 'imaging' acq. into a k-space array
image_acqs = [dset.read_acquisition(key) for key in range(dset.number_of_acquisitions())]
n_channel, n_column = image_acqs[0].data.shape
n_line = 1 + max([acq.idx.kspace_encode_step_1 for acq in image_acqs])
kspace = np.zeros([n_line, n_channel, n_column], dtype=np.complex64)
for acq in image_acqs:
    kspace[acq.idx.kspace_encode_step_1] = acq.data

image = rms_combination(ifftnd(kspace, [0,-1]))  # reconstruct image

print('k-space shape: %d lines x %d coils x %d columns'%(kspace.shape))
```

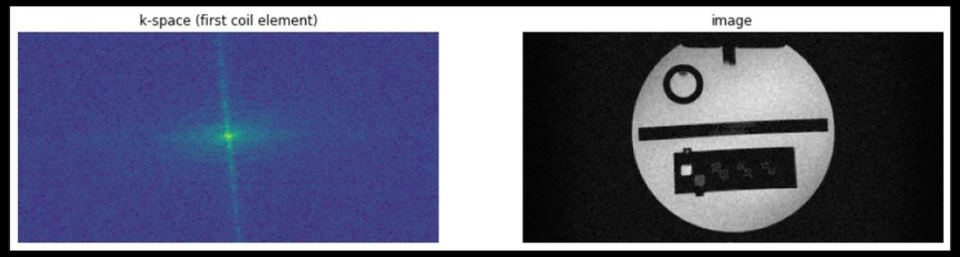
```
k-space shape: 160 lines x 2 coils x 320 columns
```



twixtools

```python
import twixtools
import numpy as np

# helper functions
def ifftnd(kspace, ax=[-1]):
    return np.fft.fftshift(np.fft.ifftn(np.fft.ifftshift(kspace, axes=ax), axes=ax), axes=ax)
def rms_combination(sig, axis=1):
    return np.sqrt(np.sum(abs(sig)**2, axis))

# parse the raw data file
twix = twixtools.read_twix('gre.dat')

# determine k-space shape and sort all 'imaging' mdbs into a k-space array
image_mdbs = [mdb for mdb in twix[-1]['mdb'] if mdb.is_image_scan()]
n_channel, n_column = image_mdbs[0].data.shape
n_line = 1 + max([mdb.cLin for mdb in image_mdbs])
kspace = np.zeros([n_line, n_channel, n_column], dtype=np.complex64)
for mdb in image_mdbs:
    kspace[mdb.cLin] = mdb.data

image = rms_combination(ifftnd(kspace, [0,-1]))  # reconstruct image

print('k-space shape: %d lines x %d coils x %d columns'%(kspace.shape))
```

```
k-space shape: 160 lines x 2 coils x 320 columns
```



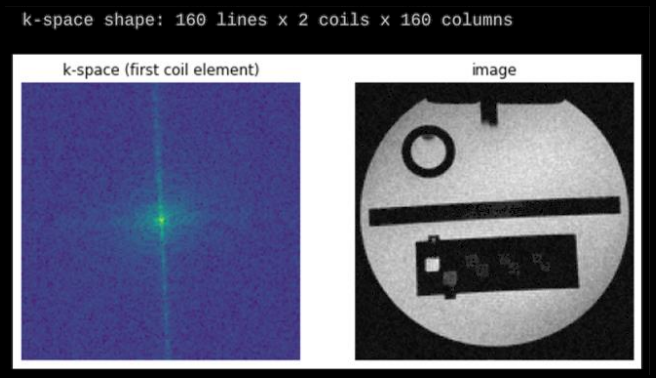
Numpy-like k-space array: map_twix

```python
import twixtools
import numpy as np

# helper functions
def ifftnd(kspace, ax=[-1]):
    return np.fft.fftshift(np.fft.ifftn(np.fft.ifftshift(kspace, axes=ax), axes=ax), axes=ax)
def rms_combination(sig, axis=1):
    return np.sqrt(np.sum(abs(sig)**2, axis))

# parse the raw data file and select the image data
twixmap = twixtools.map_twix('gre.dat')[-1]['image']

twixmap.flags['remove_os'] = True           # remove oversampling
twixmap.flags['squeeze_singletons'] = True  # remove singleton dimensions
kspace = twixmap[:]                         # read the data (array-slicin
image = rms_combination(ifftnd(kspace, [0,-1]))  # reconstruct an image

print('k-space shape: %d lines x %d coils x %d columns'%(kspace.shape))
```

Parse the file

Set flags

read data

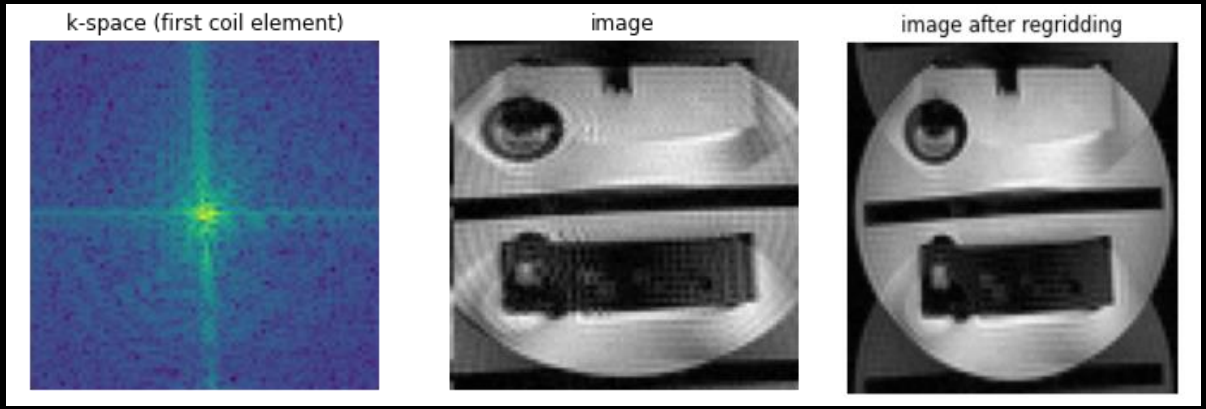


- map_twix: Open & parse file
- Optional flags determine shape of virtual array and select pre-processing steps (eg. OS-removal)
- Only requested data is read
  - Array slicing similar to numpy
  - OS-removal applied during read procedure

  → memory-friendly
- should be straightforward to adapt to MRD
  - map_mrd? (there must be a better name)

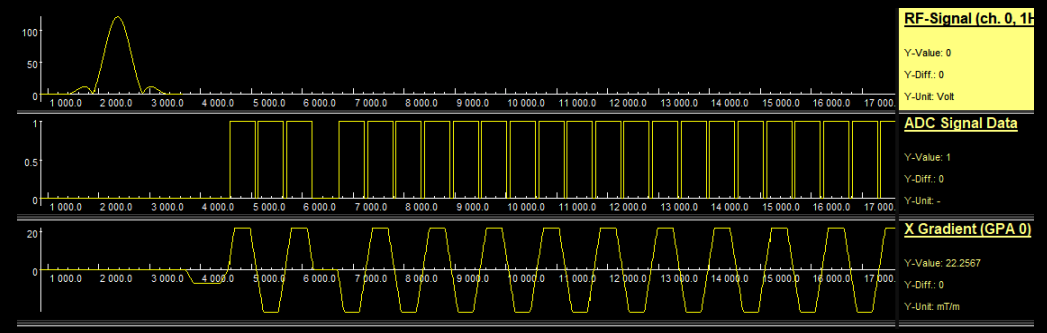
map_twix: EPI example

```
twix = twixtools.map_twix('epi.dat')[-1]          # map the twix data to twix_array objects
twix['image'].flags['remove_os'] = True           # remove oversampling
twix['image'].flags['squeeze_singletons'] = True  # remove singleton dimensions
kspace = twix['image'][:]                         # read the data
image = rms_combination(ifftnd(kspace, [0,-1]))   # reconstruct an image

twix['image'].flags['regrid'] = True  # activate regridding of ramp-sampled EPI trajectory

image_regrid = rms_combination(ifftnd(twix['image'][:], [0,-1]))  # read data & reconstruct
```



Additional features:
(simple things should be simple)

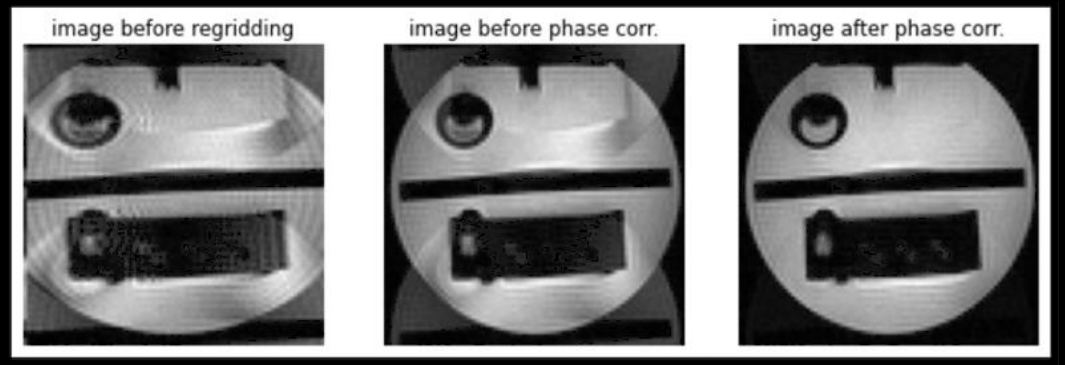
- `flags['regrid'] = True` activates regridding for EPI scans that sample during the ramps of the readout gradients.



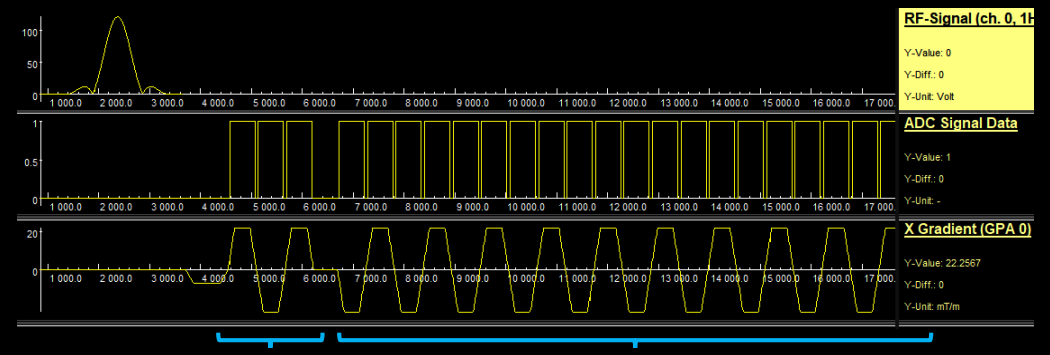
Simplified sequence diagram

map_twix: EPI example

```python
def calc_pc_corr(twixmap):
    twixmap.flags['skip_empty_lead'] = True  # do not zero-fill missing k-space lines (only k-space center present)
    twixmap.flags['average']['Seg'] = False  # do not average segment dim (indicates readout polarity)
    twixmap.flags['remove_os'] = True
    twixmap.flags['regrid'] = True
    pc = ifftnd(twixmap[:], [-1])  # ifft col dim.
    # calculate phase slope from autocorrelation and sum over coil elements (dim: -2)
    slope = np.angle((np.conj(pc[...,1:]) * pc[...,:-1]).sum(-1, keepdims=True).sum(-2, keepdims=True))
    ncol = pc.shape[-1]; x = np.arange(ncol) - ncol//2
    return np.exp(1j * slope * x)

def apply_pc_corr(data, mdh, pc_angle):
    seg = mdh['sLC']['ushSeg']  # get segment number from protocol
    pc = pc_angle[0, 0, 0, 0, 0, seg, 0, 0, 0, 0, 0, 0, 0, 0, :, :]  # we need to preserve the shape
    data = ifftnd(data, [-1])  # ifft col dim.
    data *= pc
    return data

pc_angle = calc_pc_corr(twix['phasecorr'])  # calculate phase-corr. angle
twix['image'].flags['user_func'] = [[apply_pc_corr, pc_angle]]  # add a list of user-functions with argument ("pc_angle")
kspace = twix['image'][:]
image_pc = rms_combination(ifftnd(kspace, [0]))  # reconstruct an image, fft in read already applied
```



- Ghosting artifacts from timing differences between even & odd echoes
  - Fortunately, we acquired phase-correction scans!
- no direct phase-correction support
  - However, map_twix supports user-defined functions
- Calc. correction phase and apply during read:
  `flags['user_func'] = [[func, param]]`
  `= [[apply_pc, pc_angle]]`



phase-correction scans single-shot EPI

twixtools: convert to bart cfl format

```
1  %%bash
2
3  convert_to_cfl.py gre.dat gre_converted --type image --remove_os
4
5  echo -e "\nShape of the converted file:"
6  bart show -m gre_converted
```

✓ 0.6s

```
 ---- Parsing twix file ----

Software version: VD/VE (!?)

Scan  0
100 % parsed in 0 s. Estimated 0 s remaining.

 --- Now reading image data and writing to cfl file gre_converted ---

--- Completed in 0 min and 0 s ---

Shape of the converted file:
Type: complex float
Dimensions: 16
AoD:    160     160     1       2       1       1       1       1       1
```

- Simple python-based shell interface
  - Easily extendable (argparse)
- Uses map_twix to read data into numpy array
  - Sorts data into categories, based on metadata flags
  - Access to simple pre-processing steps (e.g. oversampling removal)
- Transforms data into array order expected by bart and writes cfl file

Concluding Remarks

- Open raw data formats are important → MRD
  - Data & code sharing, reproducibility, ...
- Convenience is important
  - Interfaces, viewers
  - Toolbox integration

  → Lowers entry barrier to MRI development
- Converters for all systems (MRD!); ideally:
  - Without information loss (full protocol information)
    - Except if lossy compression is required, e.g. for long-term storage[1]
  - Should work bothways (proprietary <--> open)
    - No need to store same data twice

[1]Ehses et al. “MRI Raw Data Compression for long-term storage in large-scale population imaging”. ISMRM 2020, #1048

Projects & Code

- MRD https://ismrmrd.github.io
  - Converters for Bruker, GE, Philips, Siemens
  - HD5-based: supports wide range of languages (C/C++, python, matlab, Julia, ...)

- twixtools https://github.com/pehses/twixtools
  - Supports reading & writing of Siemens .dat files
  - Contains demo code (jupyter notebook)