université de BORDEAUX

ELECTROPHYSIOLOGY AND HEART MODELING INSTITUTE BORDEAUX

Gadgetron Bordeaux Summer School

MRD format: a necessary step for unified MR reconstruction with Gadgetron

Maxime Yon

LIRYC | Restoring the rhythm of life

Presentation 18/06/2020

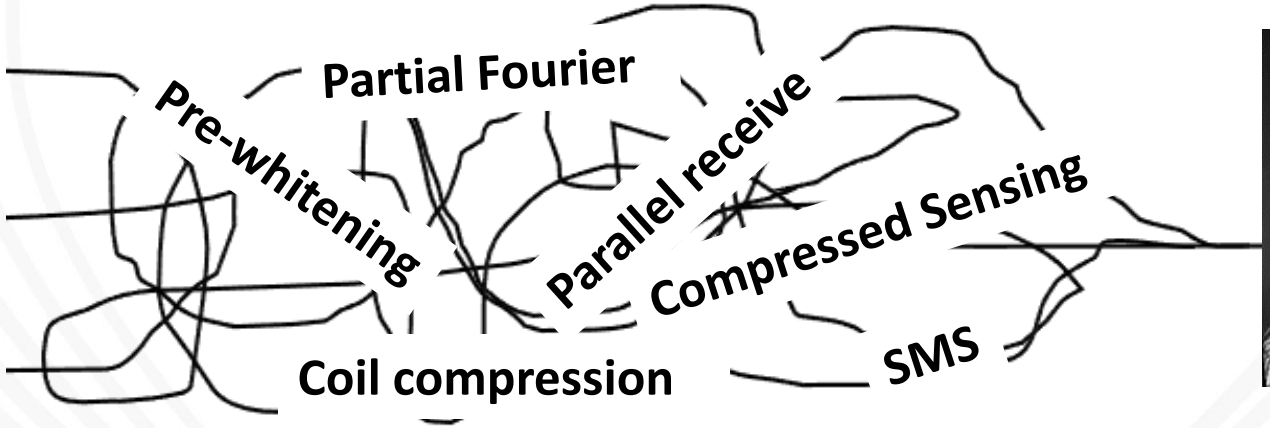
Introduction

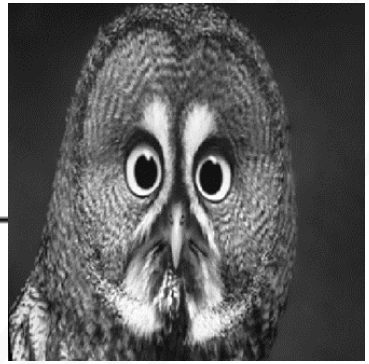
- Nowadays, each MR vendor has is own private reconstruction pipeline
- The rise of complex MR processing induces variability across vendors
- Only a unified, open source and adaptive reconstruction freeware can allow a perfect reproducibility of MR processing

Vendor-specific acquisition

Vendor-specific reconstruction

Same image ?





Gadgetron: An Open Source Framework for Medical Image Reconstruction, MRM, 2013

Introduction

- Nowadays, each MR vendor has is own private reconstruction pipeline
- The rise of complex MR processing induces variability across vendors
- Only a unified, open source and adaptive reconstruction freeware can allow a perfect reproducibility of MR processing

Vendor-specific acquisition

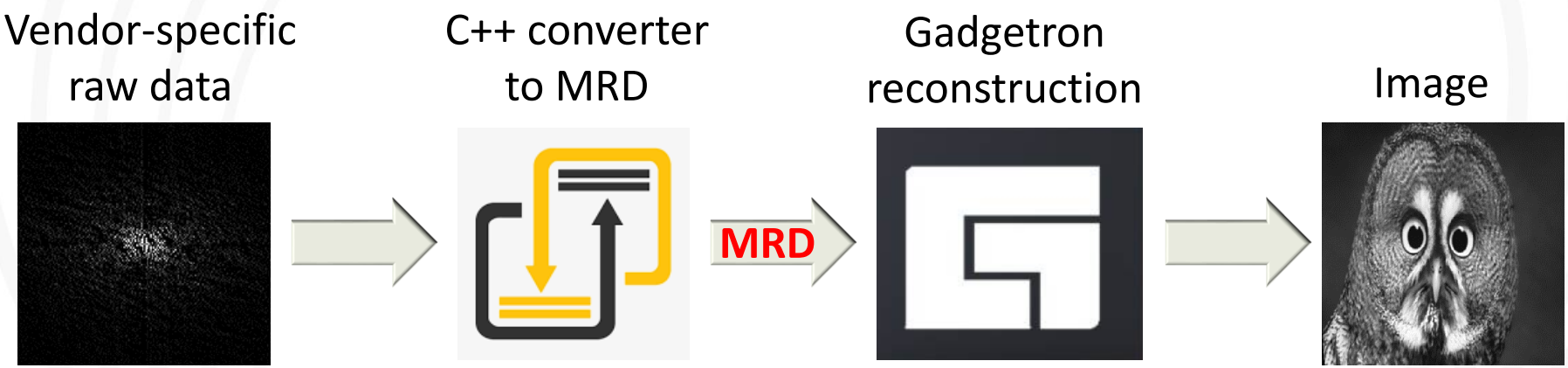
Vendor-agnostic reconstruction

Same image

Gadgetron: An Open Source Framework for Medical Image Reconstruction, MRM, 2013

Introduction

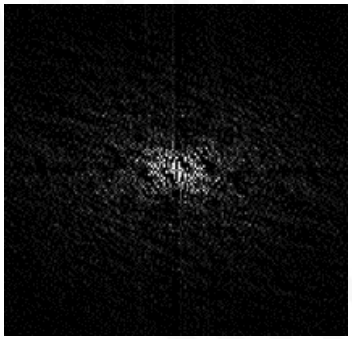
- A prerequisite for unified magnetic resonance (imaging) reconstruction is a common raw data format
- Gadgetron uses the ISMRM Raw Data format (MRD)



- Converters exist for several commercial MRI vendors: Bruker, Siemens, General Electric or Philips.

Introduction

- MRD format need to capture:
  - the raw k-space data in acquisition order before any transformation
  - the physics parameters of the data acquisition process required for image reconstruction
  - Optionally, the trajectory data or additional waveforms



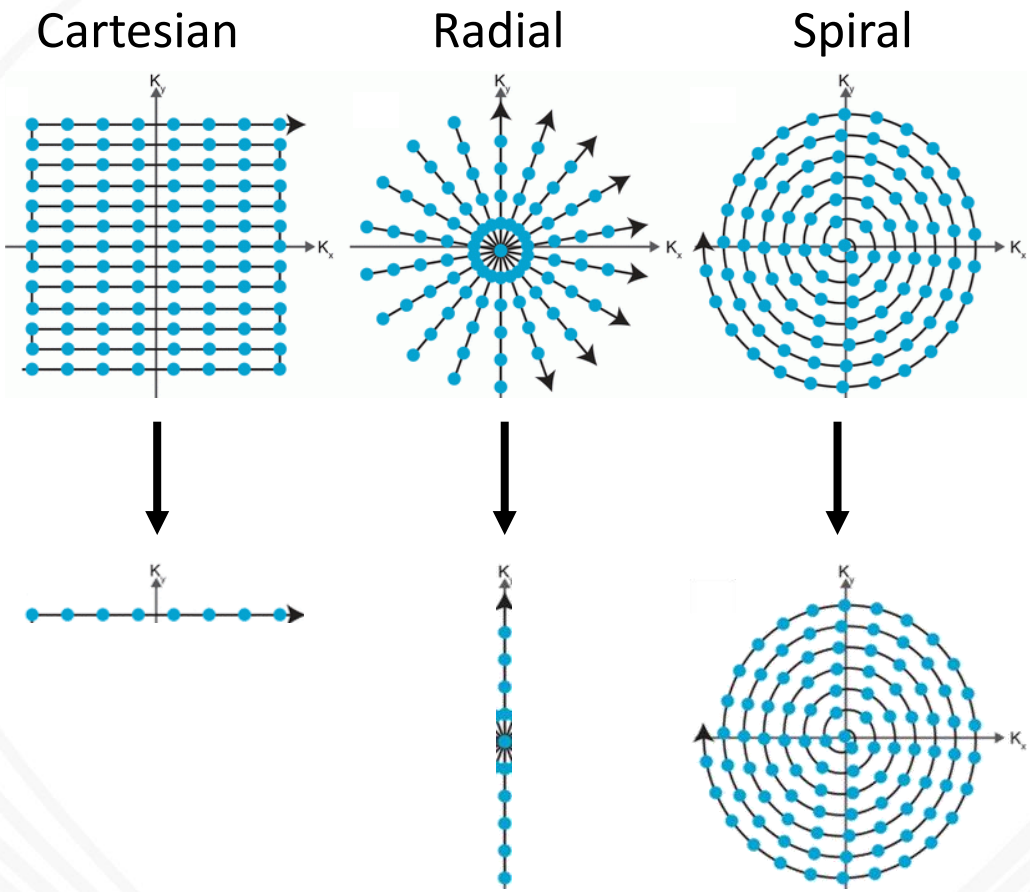
+

ISMRM Raw Data Format: A Proposed Standard for MRI Raw Datasets, MRM, 2016

I) MRD Structure

a) The MR unit

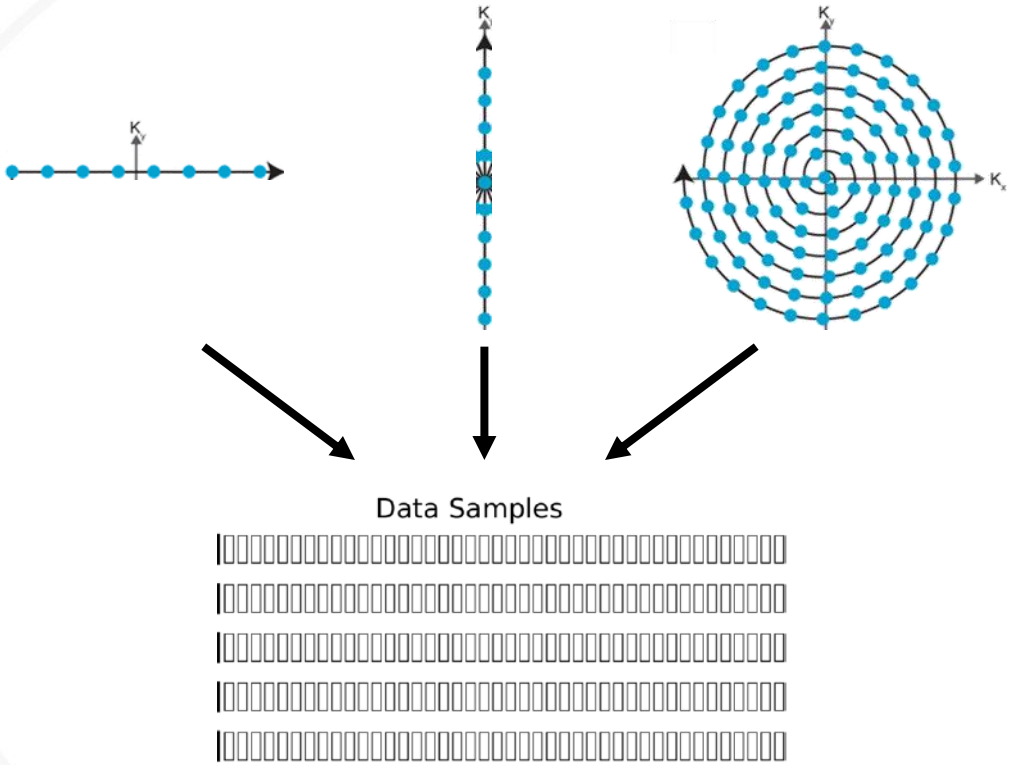
- In MR acquisition the most simple unit is the readout



I) MRD Structure

a) The MR unit

- The MRD format captures this unit avoiding any vendor specific operation

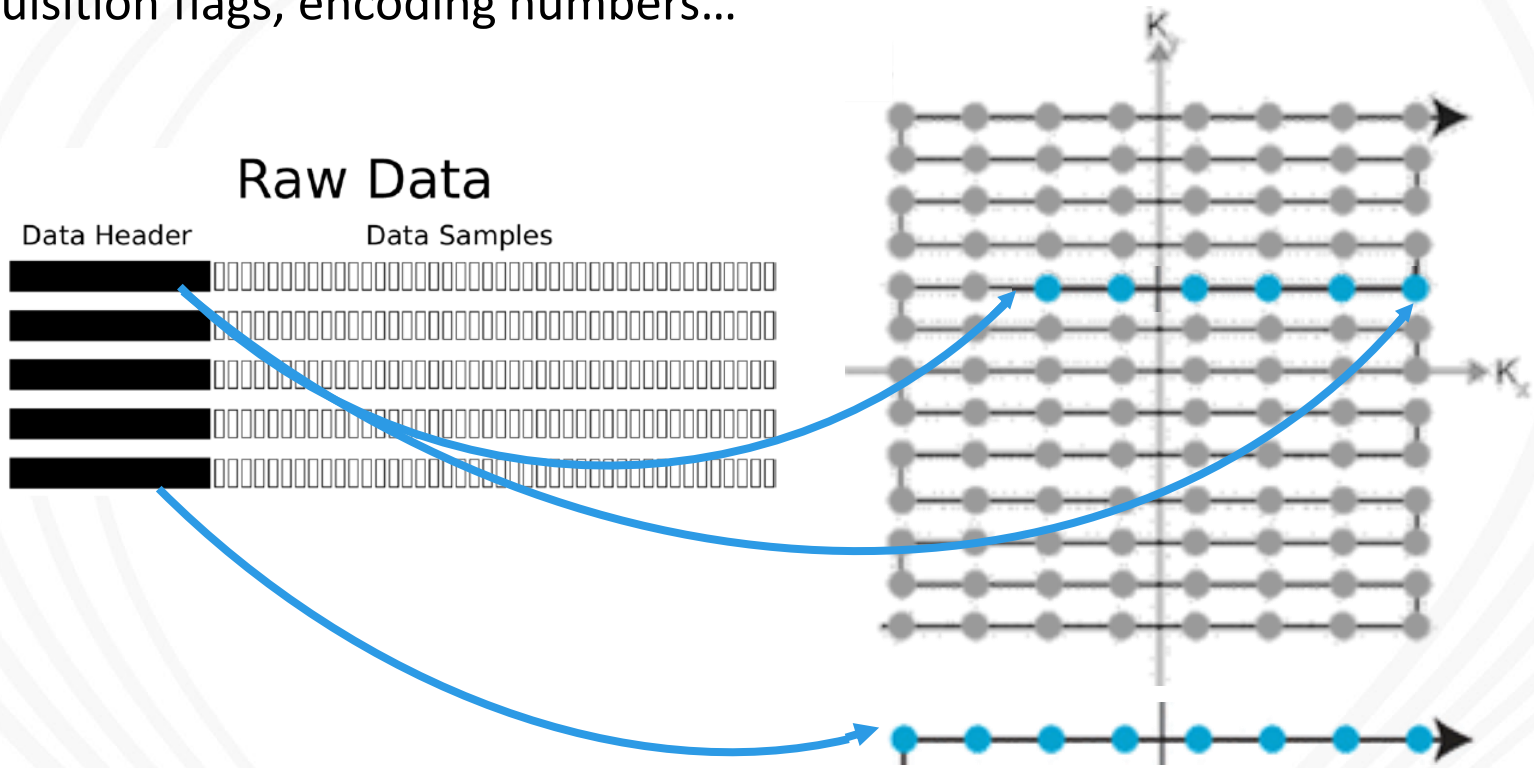


- These readout are called « data chunck »
- For multiple coils acquisition, the signal of each coil is saved

I) MRD Structure

b) The data header

- Each data item is preceded by a fixed-size Data Header including data type, acquisition flags, encoding numbers…

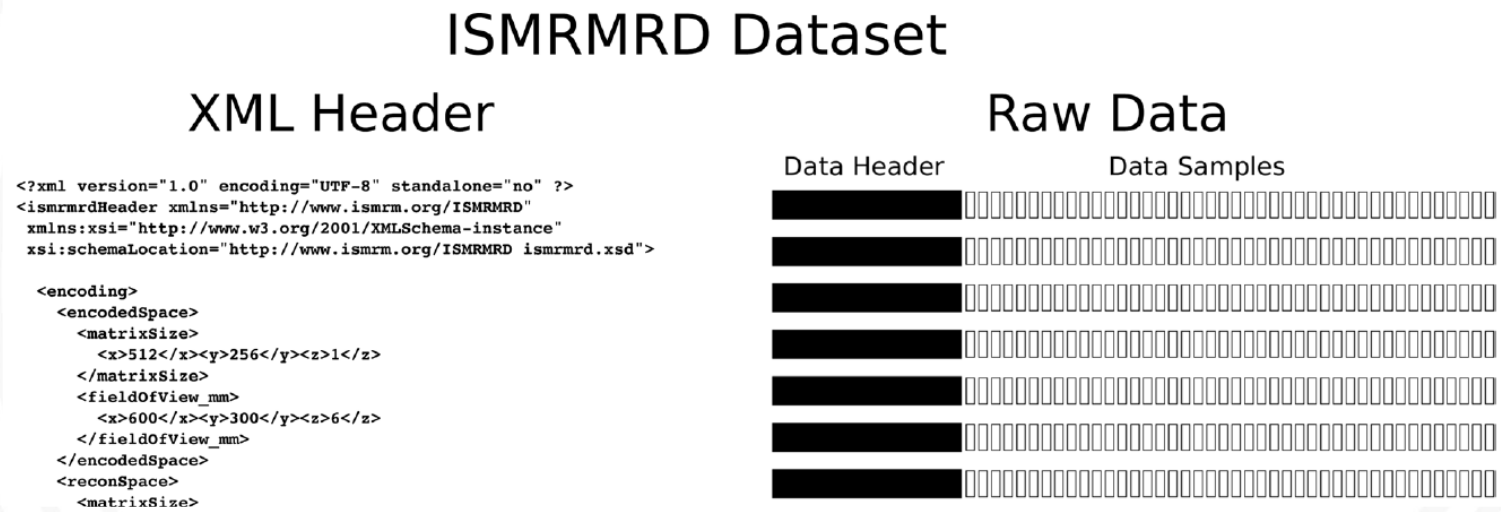


- These data chunks can have different sizes as they include navigator or calibration readout

I) MRD Structure

c) The MRD dataset also contains:

- A flexible XML Header containing an arbitrary number of fields accommodating all the parameters that may be meaningful for some experiments but not for others.



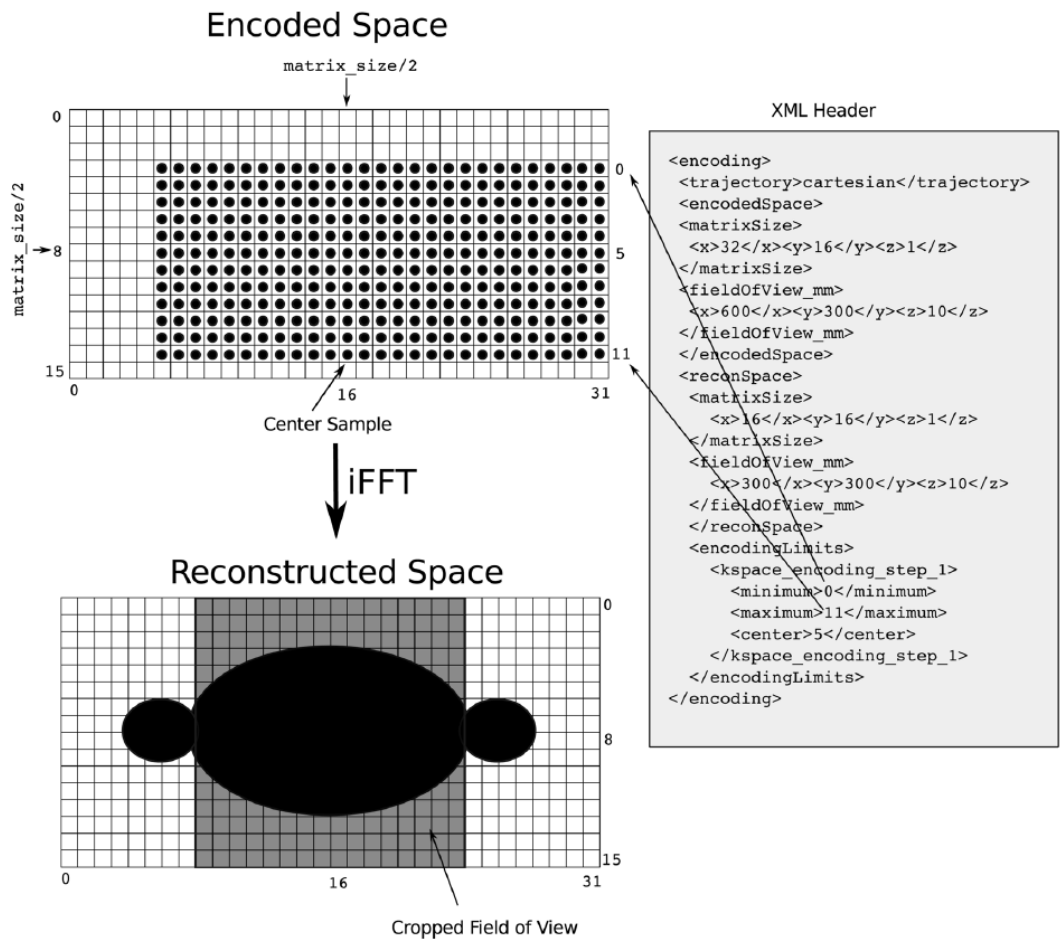
- MRD dataset = $\underbrace{\text{XML general header}}_{\text{Dataset parameters}}$ + N * ($\underbrace{\text{data\_header}}_{\text{Encoding position}}$ + $\underbrace{\text{data samples}}_{\text{Readout * coils}}$)

ISMRM Raw Data Format: A Proposed Standard for MRI Raw Datasets, MRM, 2016

I) MRD Structure

d) Encoding & recon spaces

- Key features of MRD are the encoding space and reconstruction space
- Describe the limits of the experiment
- Provides the physical size and resolution of the imaging volume and the ranges of the data header labels.

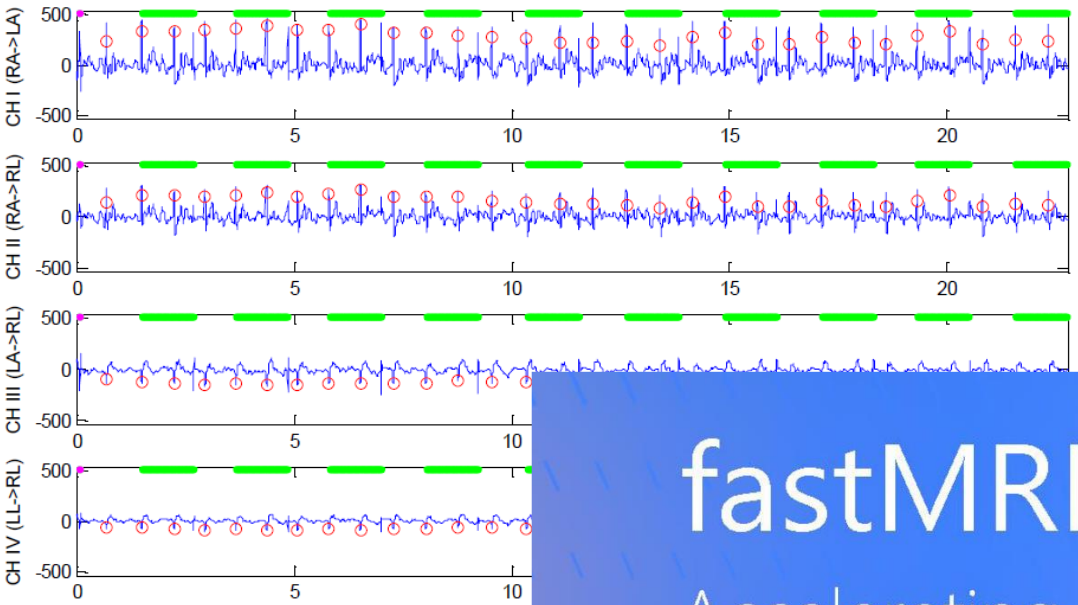


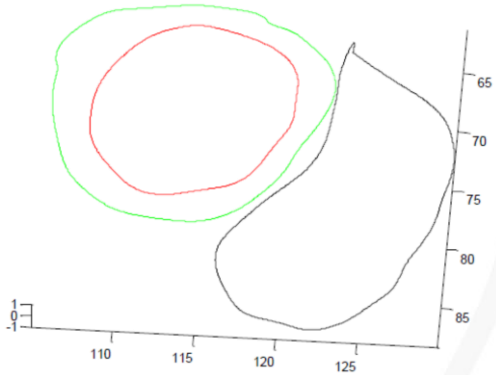
ISMRM Raw Data Format: A Proposed Standard for MRI Raw Datasets, MRM, 2016

I) MRD Structure

e) Waveforms and contours

- The MRD format also support waveform such as ECG
- It can include contours and landmark making it suitable for AI research

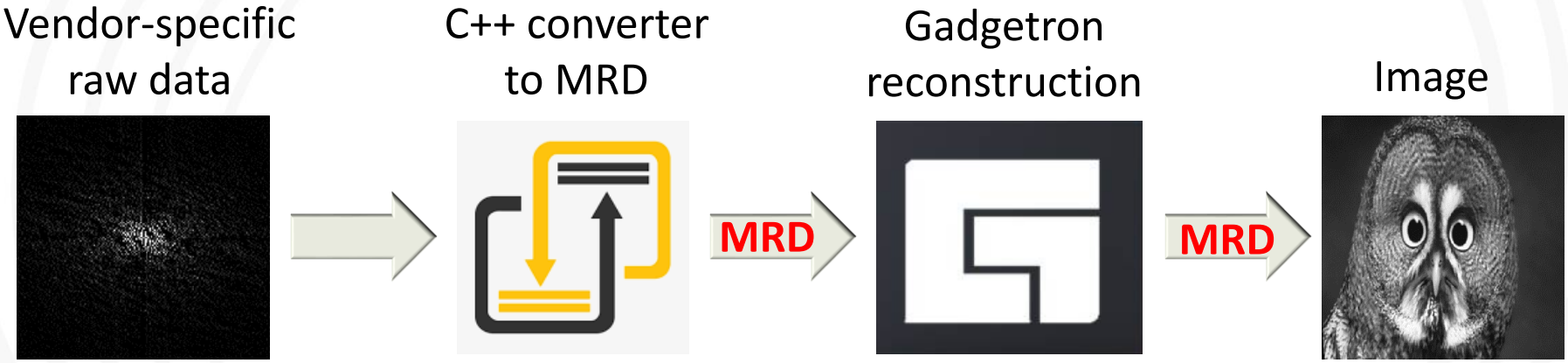




I) MRD Structure

g) Full dataset:

- MRD is also used to store images after Gadgetron reconstruction along with image header storing image and user-specified parameters



- If you prefer DICOM or NIFTI, the conversion need to be performed by the client

II) MRD library and tools

a) Library

- The ISMRMRD library provides C/C++, Python, and MATLAB (Mathworks) interfaces for reading and writing MRD files.
- The library and associated tools can be compiled on Linux, Windows, and Apple computers as described here: https://ismrmrd.github.io/
- Let's do it on Ubuntu:

```
sudo apt-get install libhdf5-serial-dev h5utils cmake cmake-curses-gui libboost-all-dev doxygen git

git clone https://github.com/ismrmrd/ismrmrd
cd ismrmrd/
mkdir build
cd build
cmake ../
make
sudo make install
```

II) MRD library and tools

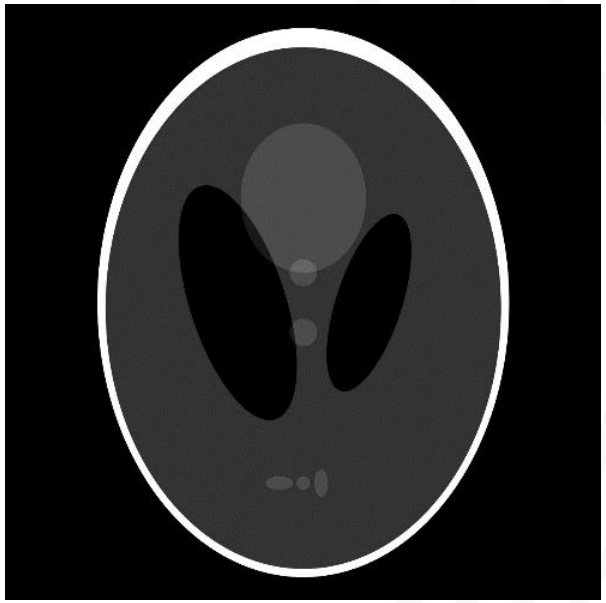
b) Phantom generation

- The MRD library can be used to create a shepp logan phantom

```
gadgetron
```

- In a new tab:

```
mkdir data
cd data
ismrmrd_generate_cartesian_shepp_logan
```



- The result is the file: testdata.h5

- Reading and writing from the HDF5 format are supported by most programming languages and computational environments such as Python or Matlab

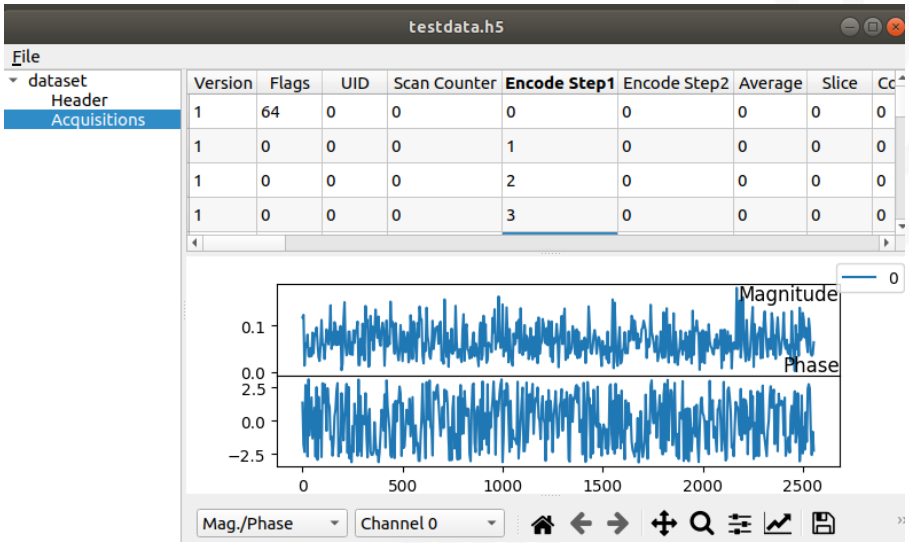
II) MRD library and tools

c) ISMRMRD viewer

- ISMRMRDviewer included in a virtual python environment is a convenient tool to explore the MRD dataset
- Let's do it on Ubuntu:

```
sudo apt-get install python3-venv
git clone https://github.com/ismrmrd/ismrmrdviewer.git
cd ismrmrdviewer/
python3 -m venv venv
. venv/bin/activate
python3 setup.py --verbose install
deactivate

. venv/bin/activate
python3 ismrmrdviewer
```



II) MRD library and tools

d) Open .h5 with Matlab

- The ISMRMRD library provides MATLAB function for reading .h5 file

```
addpath('/usr/local/share/ismrmrd/matlab')
filename = '/home/.. ../data/testdata.h5';
dset = ismrmrd.Dataset(filename, 'dataset');
header = ismrmrd.xml.deserialize(dset.readxml);
data_struct = dset.readAcquisition();
```

- 'header' is a structure containing the main XML Header fields:

```
header.version     4
header.acquisitionSystemInformation.receiverChannels     8
header.acquisitionSystemInformation.institutionName
ISMRM Synthetic Imaging Lab
header.experimentalConditions.H1resonanceFrequency_Hz    63500000
header.encoding.encodedSpace.matrixSize.x   512
header.encoding.encodedSpace.matrixSize.y   256
header.encoding.encodedSpace.matrixSize.z     1
header.encoding.encodedSpace.fieldOfView_mm.x   600
header.encoding.encodedSpace.fieldOfView_mm.y   300
...
```

II) MRD library and tools

d) Open .h5 with Matlab

- The ‘data_struct’ object contains the acquisition header, the complex data of each coils and optionally the trajectory
- The data are store in single precision complex values

data_struct

1x1 Acquisition

Property	Value
head	*1x1 AcquisitionHeader*
traj	*1x256 cell*
data	*1x256 cell*

II) MRD library and tools

d) Open .h5 with Matlab

- The 'head object' object contain the fixed-size Data Header including data type, acquisition flags, encoding numbers...
- These parameters are defined for each data chunk, that is why the second dimension here is 256

data_struct | data_struct.head

data_struct.head

Property	Value
version	1x256 uint16
flags	1x256 uint64
measurement_uid	1x256 uint32
scan_counter	1x256 uint32
acquisition_time_stamp	1x256 uint32
physiology_time_stamp	3x256 uint32
number_of_samples	1x256 uint16
available_channels	1x256 uint16
active_channels	1x256 uint16
channel_mask	16x256 uint64
discard_pre	1x256 uint16
discard_post	1x256 uint16
center_sample	1x256 uint16
encoding_space_ref	1x256 uint16
trajectory_dimensions	1x256 uint16
sample_time_us	1x256 single
position	3x256 single
read_dir	3x256 single
phase_dir	3x256 single
slice_dir	3x256 single
patient_table_position	3x256 single
idx	1x1 struct
user_int	8x256 int32
user_float	8x256 single
FLAGS	1x1 struct

II) MRD library and tools

d) Open .h5 with Matlab

- The ‘data_struct’ object contain the data chunks: 256 chunks corresponding to phase encode steps, with a readout of 512 points and 8 coils

data_struct | data_struct.data

data_struct.data

	1	2	3	4	5	6	7	8	9
1	512x8 complex single	512x8 complex single	512x8 c...	512x8 co...	512x8 co...	512x8 co...	512x8 co...	512x8 co...	512x8 co...

- The K-space matrix can be obtained by:

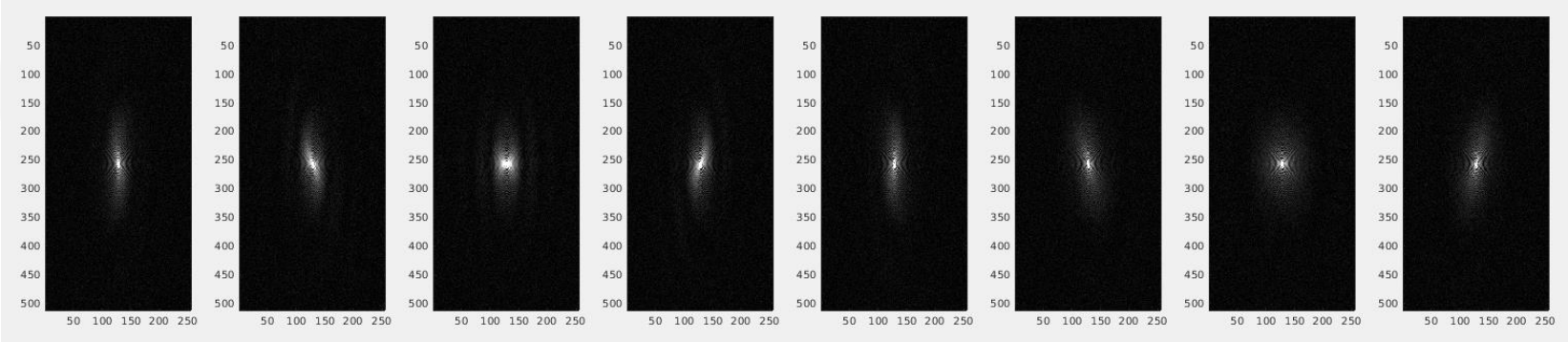
```
for ind=1:size(data_struct.data,2)
    Kspace(:,ind,:) = data_struct.data{1,ind};
end
```

- Which gives a 512 x 256 x 8 complex single matrix

II) MRD library and tools

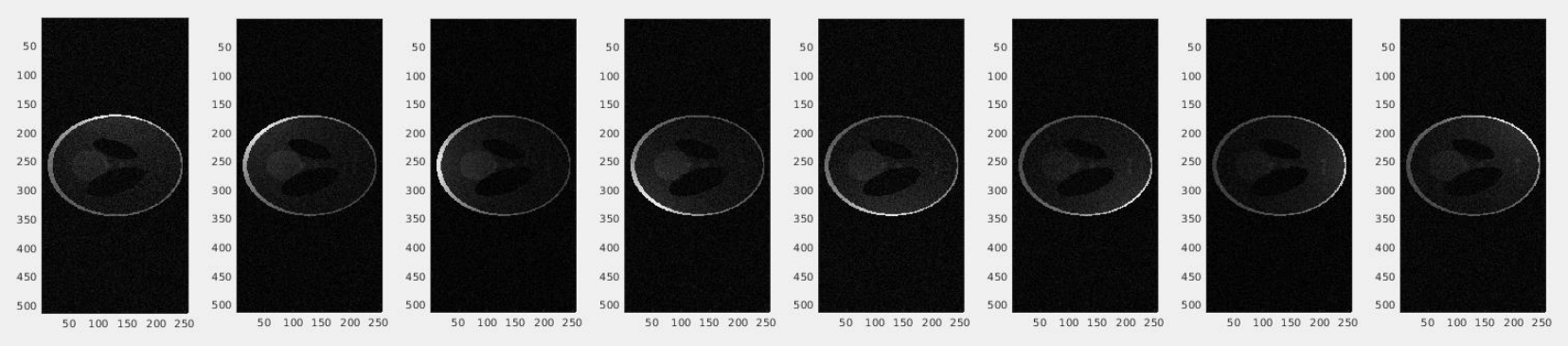
d) Open .h5 with Matlab

- This matrix contain the 8 K-spaces corresponding to the 8 coils



- Their Fourier transform give the images of each of the 8 coils

```
Images = FFTKSpace2XSpace(FFTKSpace2XSpace(Kspace,2),1);
```



II) MRD library and tools

e) Gadgetron recon and output

- Of course, this reconstruction can also be performed with Gadgetron

```
gadgetron
```

- In a new tab:

```
gadgetron_ismrmrd_client -f testdata.h5 -c default.xml -o out.h5
```

- out.h5 can also be open on Matlab with the functions h5info, h5dips and h5read and contain the reconstructed image and a basic header

```
filename = '/home/maximey/mount/maxime.yon/Data/out.h5';
hinfo = hdf5info(filename);
Img =single(h5read(filename,hinfo.GroupHierarchy.Groups(1).Groups(1).Datasets(2).Name));
header=h5read(filename_result,hinfo.GroupHierarchy.Groups(1).Groups(1).Datasets(3).Name);
```

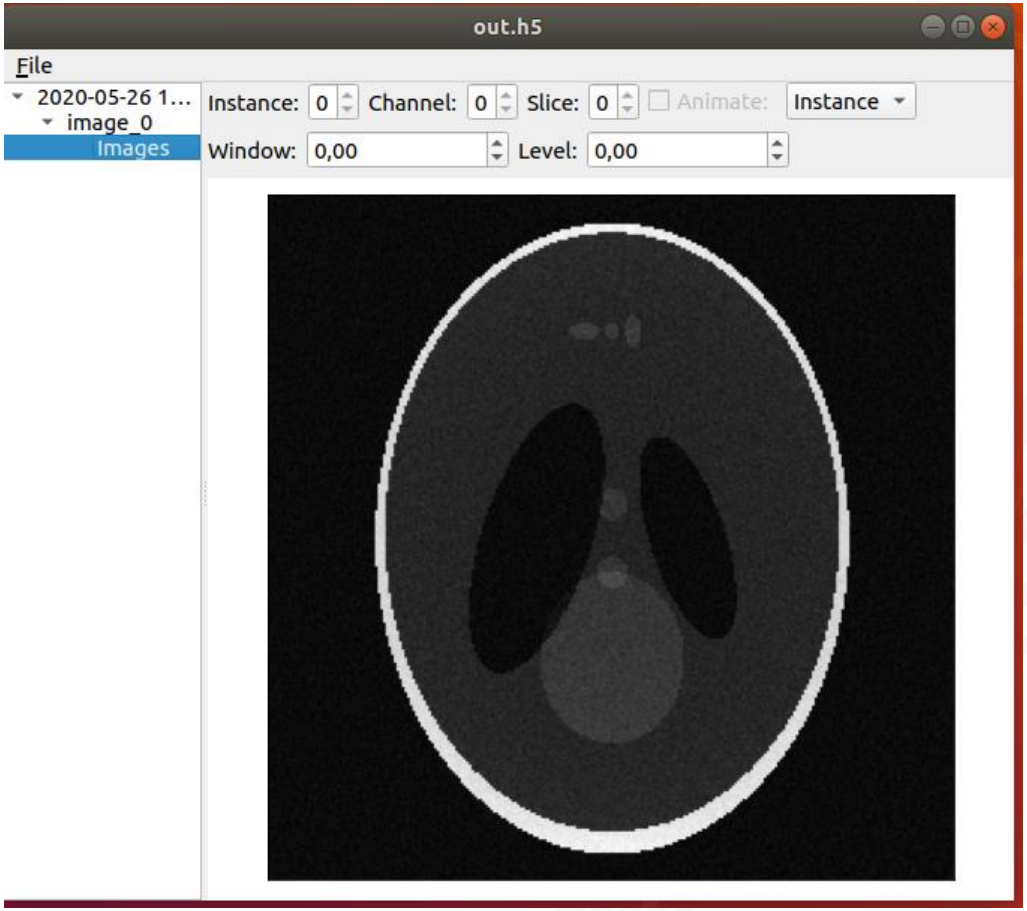
- The header is requiered if the image need to be send back to a scanner and displayed with the correct FOV, orientation…

II) MRD library and tools

e) Gadgetron recon and output

- ISMRMRDviewer can also be used to open the output image

```
cd ismrmrdviewer/
. venv/bin/activate
python3 ismrmrdviewer
```

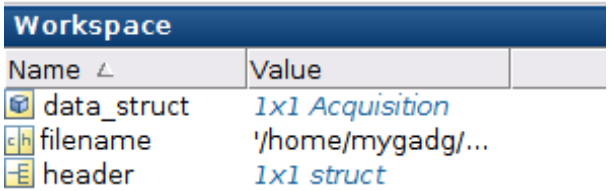


III) Hands on training

a) 2D multisclices EPI dataset

- Open the MRD EPI Dataset

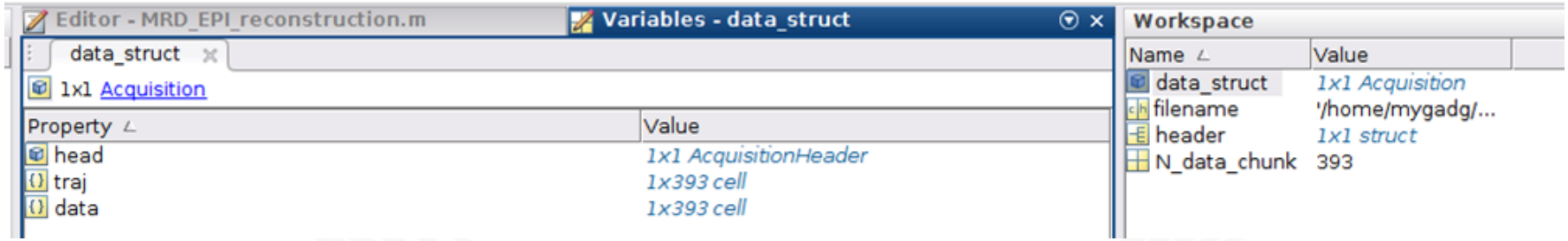
```
addpath('/usr/local/share/ismrmrd/matlab')
filename = '/home/.. ../MRD intro hands on training/FID_ep2d_se_noIPAT_3slices.h5;
dset = ismrmrd.Dataset(filename, 'dataset');
header = ismrmrd.xml.deserialize(dset.readxml);
data_struct = dset.readAcquisition();
clearvars dset;
```



- Get the number of data chunk

```
N_data_chunk = size(data_struct.data,2);
```

- Here it is 393



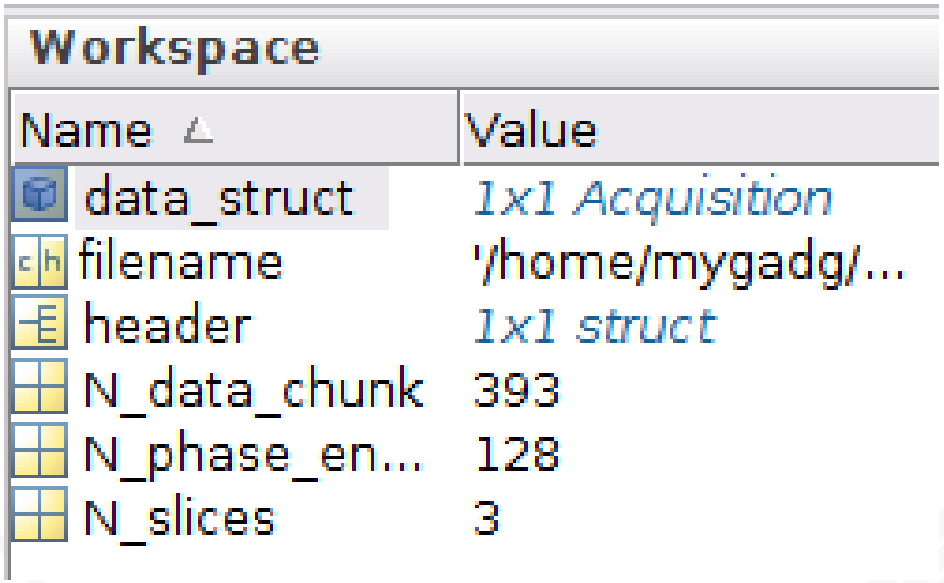
III) Hands on training

b) Exploring encoding limits

- In 2D multisclices EPI we expect these data chunk to come from each phase encode (blips) of each sclices

```
N_phase_encode = header.encoding.encodingLimits.kspace_encoding_step_1.maximum  + 1;
N_slices = header.encoding.encodingLimits.slice.maximum + 1;
```

- 128 x 3 = 384, we have 9 additional data chunk



III) Hands on training

b) Exploring encoding limits

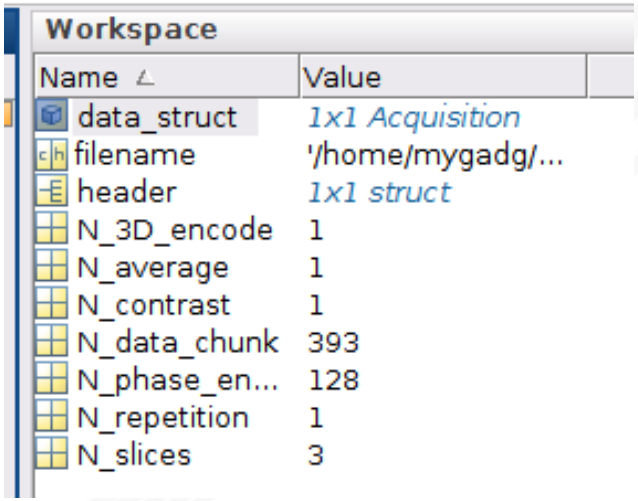
- In 2D multisclices EPI we expect these data chunk to come from each phase encode (blips) of each sclices

```
N_phase_encode = header.encoding.encodingLimits.kspace_encoding_step_1.maximum  + 1;
N_slices = header.encoding.encodingLimits.slice.maximum + 1;
```

- 128 x 3 = 384, we have 9 additional data chunk

```
N_3D_encode = header.encoding.encodingLimits.kspace_encoding_step_2.maximum  + 1;
N_average = header.encoding.encodingLimits.average.maximum + 1;
N_repetition = header.encoding.encodingLimits.repetition.maximum + 1;
N_contrast = header.encoding.encodingLimits.contrast.maximum + 1;
```

- Not coming from additional encoding steps



III) Hands on training

c) Looking at Flags

- The additional data chunk can also come from parallel calibration, navigation data or phase correction data

```
parallel_calibration = data_struct.head.flagIsSet('ACQ_IS_PARALLEL_CALIBRATION');
navigation_data = data_struct.head.flagIsSet('ACQ_IS_NAVIGATION_DATA');
phase_corr_data = data_struct.head.flagIsSet('ACQ_IS_PHASECORR_DATA');

N_parallel_calibration = sum(parallel_calibration);
N_navigation_data = sum(navigation_data);
N_phase_corr_data = sum(phase_corr_data);
```

- The 9 additional data chunk are phase correction readout used for ghost correction in EPI processing

Workspace

Name	Value
data_struct	*1x1 Acquisition*
filename	'/home/mygadg/...
header	*1x1 struct*
N_data_chunk	393
N_navigation_data	0
N_parallel_calibration	0
N_phase_corr_data	9
navigation_data	*1x393 double*
parallel_calibration	*1x393 double*
phase_corr_data	*1x393 double*

III) Hands on training

d) Create kspace matrix

- Here we will simply discard them and create a matrix with the image readout only

```
image_data = find(parallel_calibration+navigation_data+phase_corr_data==0);

readout_size = size(data_struct.data{1,image_data(1,1)},1);
N_coils = size(data_struct.data{1,image_data(1,1)},2);
kspace = zeros(readout_size, N_phase_encode, N_coils,N_slices);

for ind = image_data
      kspace(:,data_struct.head.idx.kspace_encode_step_1(1,ind)+1,:,data_struct.head
      .idx.slice(1,ind)+1) = data_struct.data{1,ind};
end
```

- The size of the kspace matrix is:
  256 readout points
  128 phase encode
  26 coils
  3 slices

Command Window

```
K>> size(kspace)

ans =

   256   128    26     3
```

III) Hands on training

e) is reversed flag

- In EPI the odd and even echoes are acquired in opposite direction

```
is_reversed_acq = data_struct.head.flagIsSet('ACQ_IS_REVERSE');
is_reversed_acq = is_reversed_acq(image_data);

kspace_flip = kspace;
kspace_flip (:,is_reversed_acq+1,:,:) = flip(kspace_flip (:,is_reversed_acq+1,:,:),1);
```

- This information is stored in the `'ACQ_IS_REVERSE'` Flag and can be used to flip the readout

Editor - MRD_EPI_reconstruction.m* | Variables - is_reversed_acq

is_reversed_acq

1x384 double

	1	2	3	4	5	6	7	8
1	0	1	0	1	0	1	0	1
2								

III) Hands on training

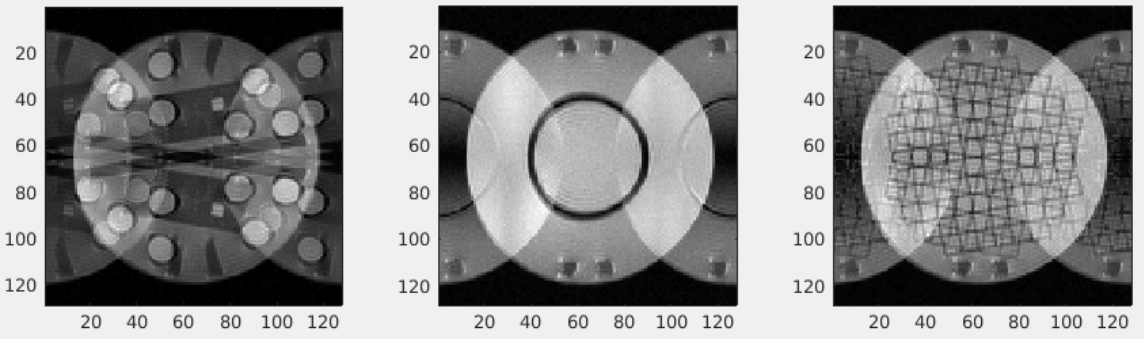
f) Fourier transform and oversampling

- In EPI the odd and even echoes are acquired in opposite direction

```
image_noRegrid = FFTKSpace2XSpace(FFTKSpace2XSpace(kspace_flip,1),2);
image_noRegrid = squeeze(sum(abs(image_noRegrid),3));

if size(image_noRegrid,1)/header.encoding.encodedSpace.matrixSize.x(1,1)==2
   image_noRegrid = image_noRegrid(round(size(image_noRegrid,1).*0.25)+1:
   round(size(image_noRegrid,1).*0.75),:,:);
end

figure()
subplot(1,3,1)
imagesc(image_noRegrid(:,:,1));
axis image
subplot(1,3,2)
imagesc(image_noRegrid(:,:,2));
axis image
subplot(1,3,3)
imagesc(image_noRegrid(:,:,3));
axis image
colormap gray
```



III) Hands on training

g) Regridding

- Readout acquisition is performed during the ramp time
- Gradient shape parameters are stored in user parameters

```
for ind = 1:size(header.encoding.trajectoryDescription.userParameterLong,2)
    parameters.(header.encoding.trajectoryDescription.userParameterLong(1,ind).name) =
header.encoding.trajectoryDescription.userParameterLong(1,ind).value;
end

for ind = 1:size(header.encoding.trajectoryDescription.userParameterDouble,2)
    parameters.(header.encoding.trajectoryDescription.userParameterDouble(1,ind).name)
= header.encoding.trajectoryDescription.userParameterDouble(1,ind).value;
end
parameters.readout = readout_size;
parameters.N_phase_encode = N_phase_encode;
parameters.N_phase_recon = header.encoding.reconSpace.matrixSize.x(1,1);
parameters.N_slices = N_slices;
parameters.position = data_struct.head.position(:,image_data(1,1));
parameters.read_dir = data_struct.head.read_dir(:,image_data(1,1));
parameters.FOV_1 = header.encoding.encodedSpace.fieldOfView_mm.x(1,1);
parameters.is_reversed_acq = is_reversed_acq;
```

III) Hands on training

g) Regridding

- RampUpTime, rampDownTime, flatTopTime are used to perform the regridding
- The is reversed flag and readout size will also be used for additional processing

parameters

1x1 struct with 16 fields

Field	Value
etl	128
numberOfNavigators	3
rampUpTime	160
rampDownTime	160
flatTopTime	370
acqDelayTime	102
numSamples	256
dwellTime	1.9000
readout	256
N_phase_encode	128
N_phase_recon	128
N_slices	3
position	[19.6126;-12.5666;-6.5375]
read_dir	[0;5.9605e-08;-1.0000]
FOV_1	250
is_reversed_acq	*1x384 double*

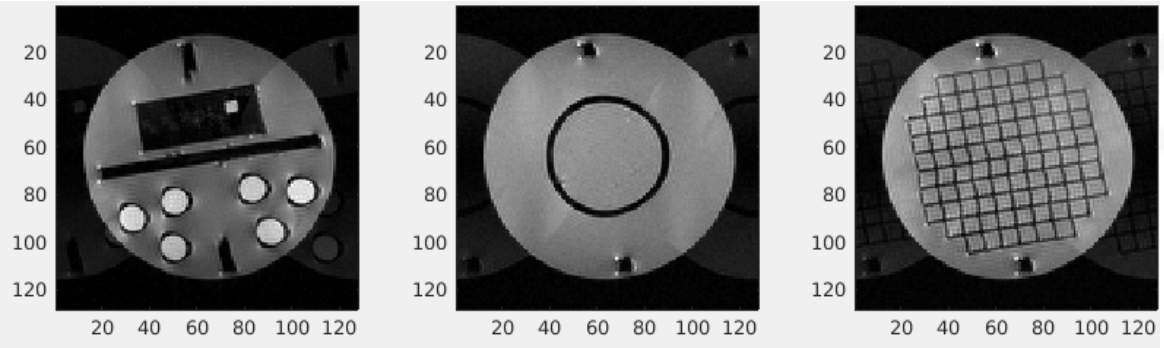
III) Hands on training

g) Regridding

- The function: **`EPI_trapezoid_regridding`** perform regridding, Fourier transform in dimension 1 and remove oversampling
- This allows to decrease ghost artefacts

```
[kspace_corr] = EPI_trapezoid_regridding(parameters,kspace);
image_Regrid = FFTKSpace2XSpace(kspace_corr,2);
image_Regrid = squeeze(sum(abs(image_Regrid),3));

figure()
subplot(1,3,1)
imagesc(image_Regrid(:,:,1));
axis image
subplot(1,3,2)
imagesc(image_Regrid(:,:,2));
axis image
subplot(1,3,3)
imagesc(image_Regrid(:,:,3));
axis image
colormap gray
```



Conclusion

- MRD is a versatile format for storing raw MR data and the parameters required for reconstruction
- It support large datasets
- The HDF file format allows easy interfacing with C/C++ python or Matlab reconstruction code
- ISMRMRD library provides tools for opening or viewing MRD datasets