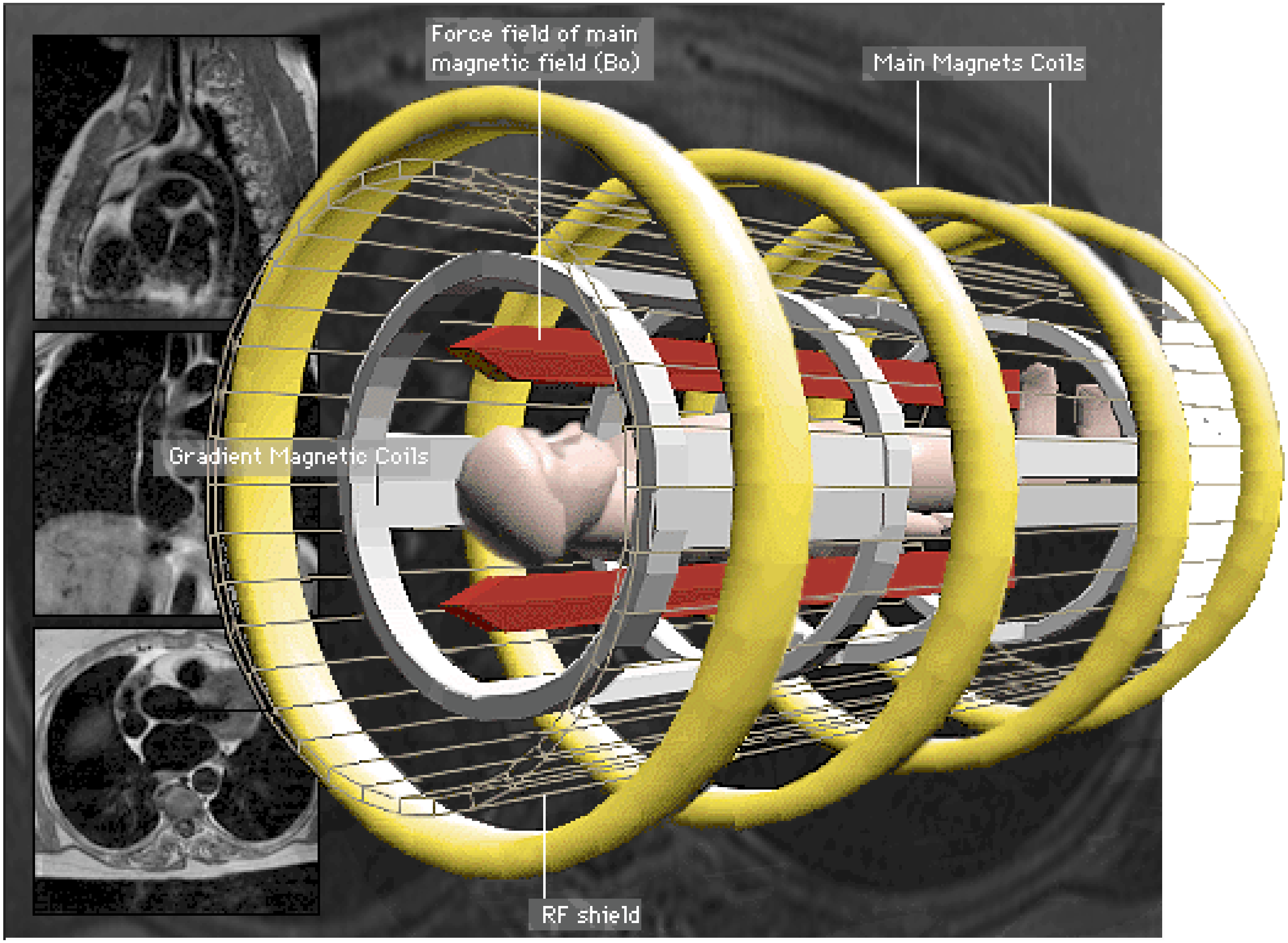
Python Imaging Tools for Reconstructing Magnetic Resonance Images

Mike Trumpis, Daniel Sheltraw, Jarrod Millman, and Mark D'Esposito

UC Berkeley Brain Imaging Center

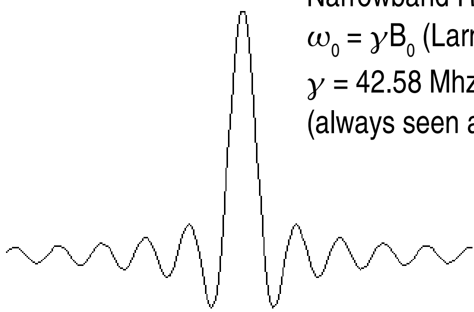
Schematic MRI

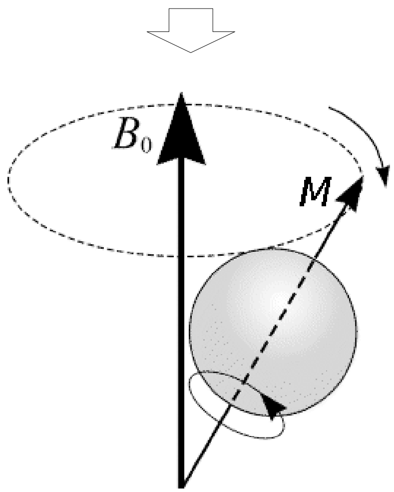


Artwork courtesy of Rebecca Cagle, National Library of Medicine-Lister Hill Center for Biocommunication

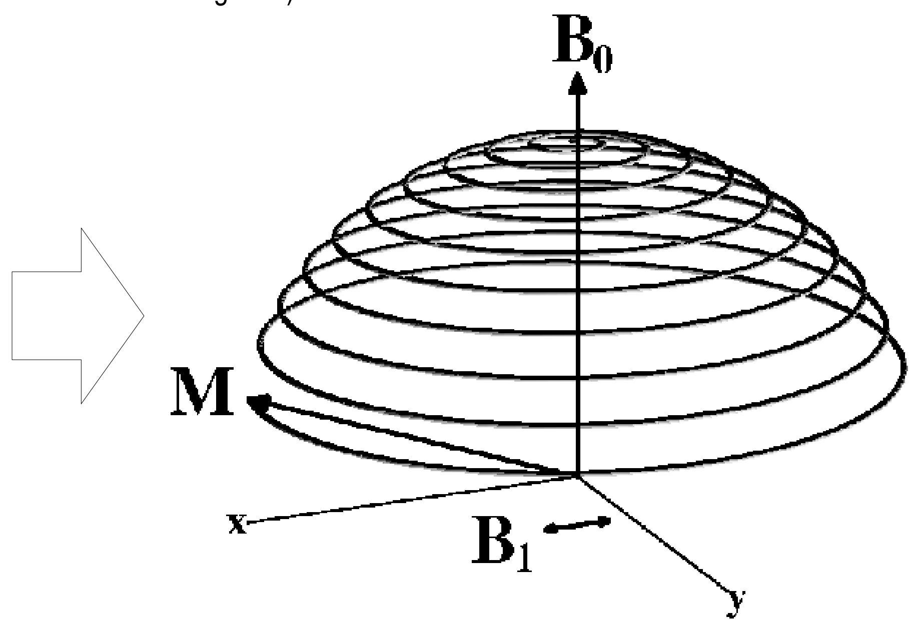
Precession of tipped magnitization vector

Narrowband RF pulse with center frequency
$\omega_0 = \gamma B_0$ (Larmor frequency)
γ = 42.58 Mhz/T = gyromagnetic ratio for ^{1}H
(always seen as a sinc function in diagrams)





An "aligned" proton in the large static field B_0--slightly non-parallel due to spin. Rotation about B_0 also at Larmor frequency.



After applying the low power RF pulse in the B_1 direction, the magnetization vector is tipped away from B_0 and begins to precess back to "equilibrium"

The movement of the magnetization vector in the x-y plane is the source of the MR signal

MR Signal and k-space

$$S(t) = \frac{i\omega_o}{2}\int M_+(\mathbf{r}, t_o) e^{i\gamma\mathbf{r}\cdot\int_{t_o}^{t}\mathbf{G}(t')dt'} d^3\mathbf{r}.$$

Definition of the MR signal

Expressing this signal in terms of a reciprocal space: k-space

Conventionally, we define our image to be the initial state of $M_+(\mathbf{r})$, and create a function $\mathbf{k}(t)$:

$$\mathbf{k}(t) = \frac{\gamma}{2\pi}\int_{t_o}^{t}\mathbf{G}(t')dt' \Rightarrow S(\mathbf{k}(t)) = \int I(\mathbf{r}) e^{i2\pi\mathbf{r}\cdot\mathbf{k}(t)} d^3\mathbf{r}.$$

Now it is clear that this signal can be interpreted as an image "transformed" into k-space

$$I(\mathbf{r}) = \int S(\mathbf{k}) e^{-i2\pi\mathbf{r}\cdot\mathbf{k}} d^3\mathbf{k}.$$

Berkeley
University of California

Image distortions in EPI (echo-planar imaging)

An EPI scan records the whole volume in one T1 period

Good: meets the time-resolution required for fMRI studies

Bad: phase related artifacts get worse as time goes on

Two main artifacts related to EPI scanning and reconstruction:

Nyquist (N/2) ghosting (a timing/sampling based error)

Field inhomogeneity induced geometric distortion (a breakdown of linearity of the transform)

Typically, correction efforts single out one artifact or the other...

We try to model their effects simultaneously.

Unified distortion correction with an operation kernel

A model of the perturbation kernel, with system and object related phase errors

$$F_s[r;m,m']=e^{i[\Delta\omega_0+\boldsymbol{r}\cdot\delta\tilde{\boldsymbol{g}}^0]t_m}e^{i(-1)^m g_r^0\boldsymbol{r}\dot{\delta}\boldsymbol{k}}\sum_p e^{i\pi(m'-m)p/N}e^{i\gamma\phi_{rps}mT_l}$$

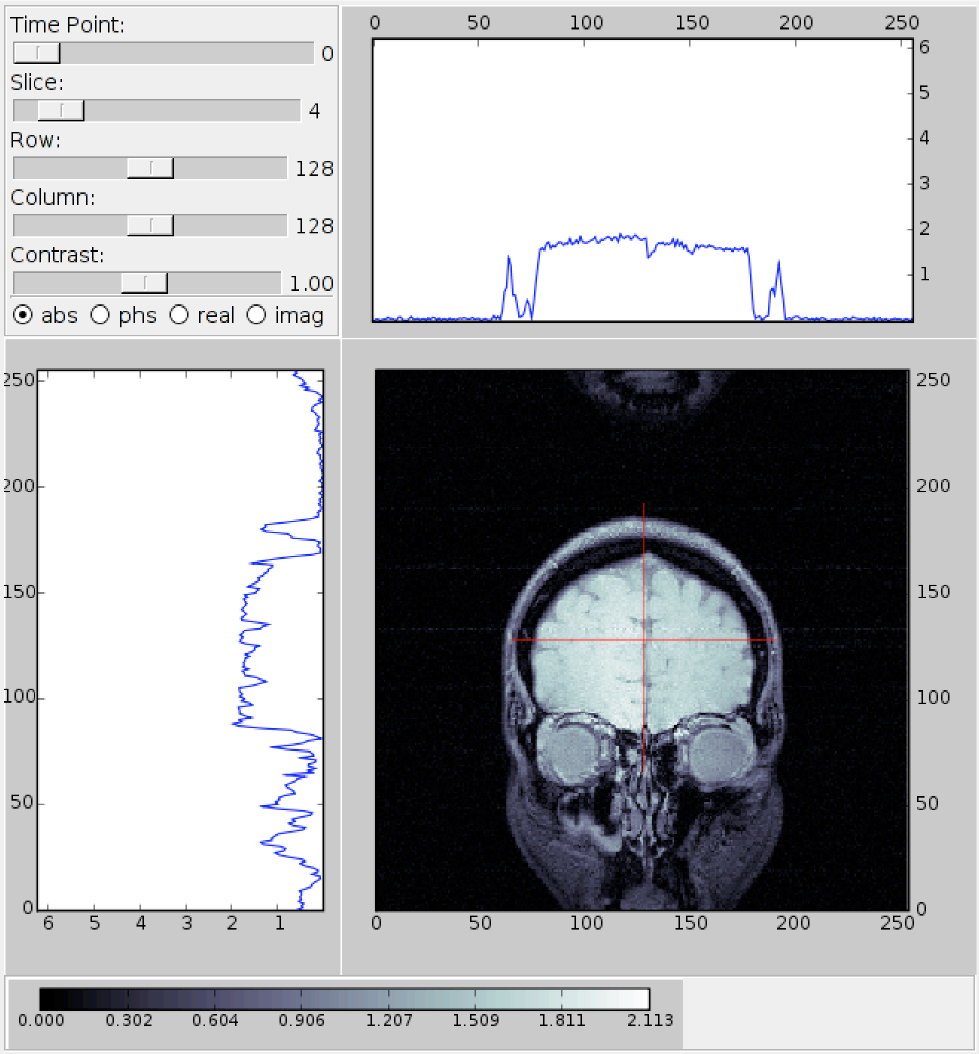
phase shifters, account for global shifting and shearing

point remapping function, ruins the linearity assumed by the Fourier transform

The MR signal modified to include the effects of the perturbation kernel

$$\hat{S}_s[r;m]=e^{i[\Delta\omega_0+\boldsymbol{r}\cdot\delta\boldsymbol{g}^0]t_m}e^{i(-1)^m g_r^0\boldsymbol{r}\dot{\delta}\boldsymbol{k}}\sum_{pm'}\hat{S}_s[r;m']e^{i\pi(m'-m)p/N}e^{i\gamma\phi_{rps}mT_l}$$

MRI reconstruction with Python Imaging Tools



Sliceviewer showing a coronal GEMS slice

Signal processing steps are modularized and utilize objects naturally related to the problem at hand

Provides specialized corrections for EPI-specific distortions

New imaging operation modules can literally be “dropped in”

Reconstructs many forms of data (anatomicals and functionals)

Written in Python and Numeric/Matplotlib!

Fundamental classes of Imaging Tools

Imaging Tools is built up around abstractions of the elements of our imaging problem.

BaseImage is the generic image type.
Every BaseImage at least has *data* and dimension information (*ndim, tdim, zdim, ydim, xdim*).

Common images are **FidImage** and **AnalyzeImage**

An **Operation** is an object that may contain one or more **Parameter** objects and implements a method `run(self, image)`. It is expected to receive a **BaseImage** and perform some transformation to its data.

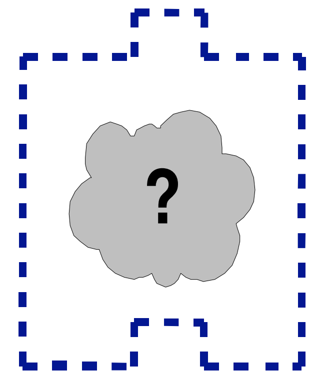
Examples of **Operations** are **InverseFFT** and **WriteImage**

Not a class, but ...

An ***oplist*** includes a sequence of **Operation** names (and any **Parameter** specifications) to take the image through.

```
[ReorderSlices]
flip_slices=False

[UnbalPhaseCorrection]

[GeometricUndistortionK]
fmap_file = fieldmap-0
mask_file = volmask-0

[FermiFilter]
trans_width=0.3
cutoff=0.95

[InverseFFT]

[ViewImage]
```

Configuring Operations with an Oplist



What is an **Operation**?

- is initialized with a (possibly empty) dictionary
- has a (possibly empty) tuple of **Parameter** objects whose names key the dictionary
- overloads `run(self, image)`

```
[ReorderSlices]
flip_slices=False

[UnbalPhaseCorrection]

[GeometricUndistortionK]
fmap_file = fieldmap-0
mask_file = volmask-0

[FermiFilter]
trans_width=0.3
cutoff=0.95

[InverseFFT]

[ViewImage]
```

What is a **Parameter**?

- specifies the *name* and *type* of an expected parameter to an operation
- “valuates” the string-typed parameter value into correct type
- provides a default value if none is given in the oplist

*how **Parameter** information is fed into an **Operation** instance:*

```
def configure(self, **kwargs):
    for p in self.params:
        self.__dict__[p.name] = p.valuate(kwargs.pop(p.name, p.default))
```

Berkeley
University of California

Python Imaging Tools basics: Setup of Recon

oplist.txt

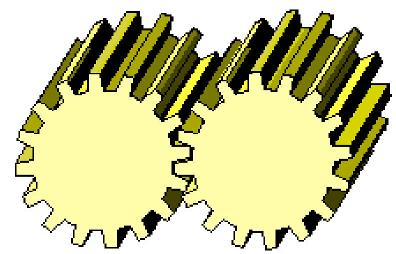
OrderedConfigParser

```
ZeroPad
params{...}

InverseFFT
params{...}

WriteImage
params{...}
```

(Opclass, args) pairs:

```
[(<class ZeroPad ...>, {params}),
 (<class InverseFFT ...>, {params}),
 (<class WriteImage ...>, {params})]
```

```
[ op(**args) for op, args in options.operations]
```



```
[<imaging.operations.ZeroPad.ZeroPad object at 0xb73c9dac>,
<imaging.operations.InverseFFT.InverseFFT object at 0xb73c9b4c>,
<imaging.operations.WriteImage.WriteImage object at 0xb73c998c>]
```

{Opname: opclass} mapping

All available operations

Ordered sequence of objects, ready to receive call to run(image)

CIRL

Berkeley
University of California

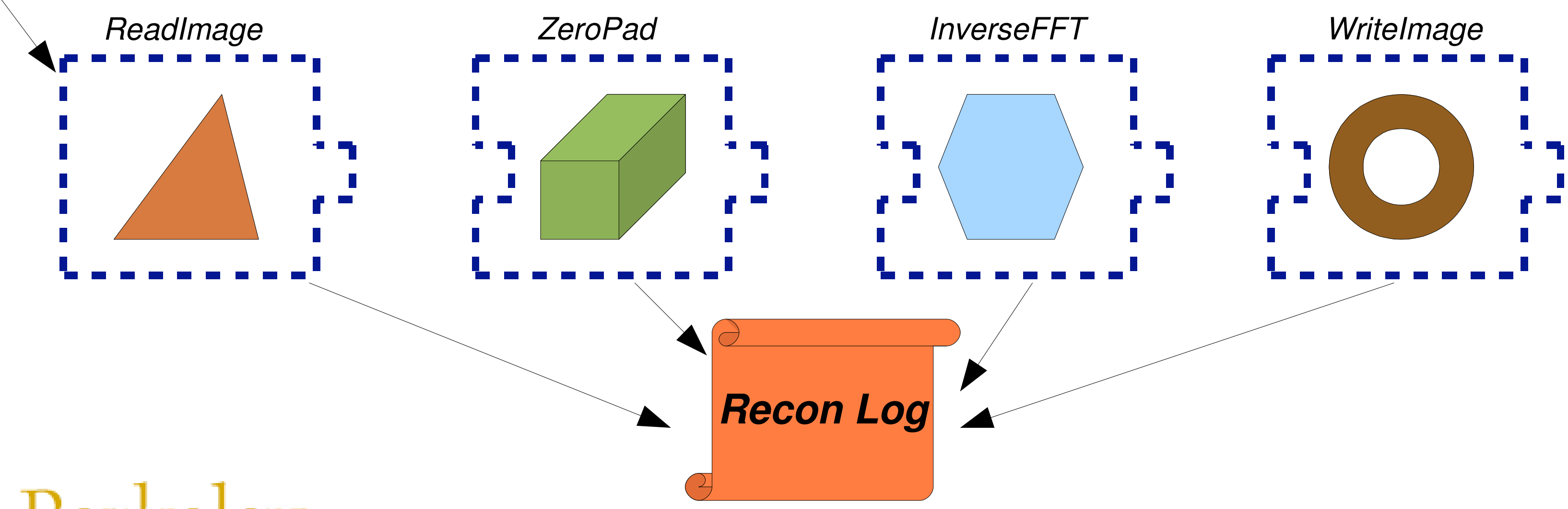
Python Imaging Tools basics: Recon's operation pipeline

"Ordered sequence of objects, ready to receive call to run(image)"

```
[<imaging.operations.ZeroPad.ZeroPad object at 0xb73c9dac>,
<imaging.operations.InverseFFT.InverseFFT object at 0xb73c9b4c>,
<imaging.operations.WriteImage.WriteImage object at 0xb73c998c>]
```

```
def runOperations(self, operations, image, runlogger):
    for operation in operations:
        operation.run(image)
        runlogger.logop(operation)
```

Typically, ReadImage is configured from command-line statements, and run explicitly by the Recon system



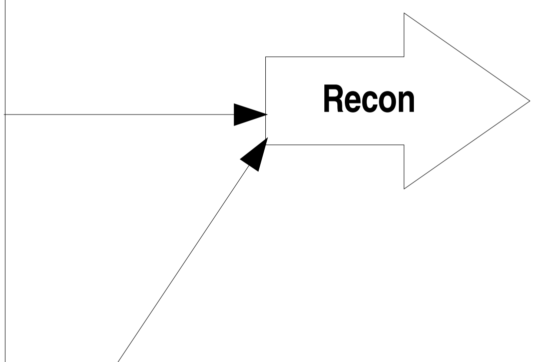
Provenance Tracking: the Recon log file

To facilitate reproducible data, Recon generates a log file which is itself an executable script.

Calling that script from the command line runs the same raw data through an identical battery of operations.

```
[ReorderSlices]

[UnbalPhaseCorrection]
lin_radius = 70

[GeometricUndistortionK]

[InverseFFT]

[ViewImage]
```

original oplist file

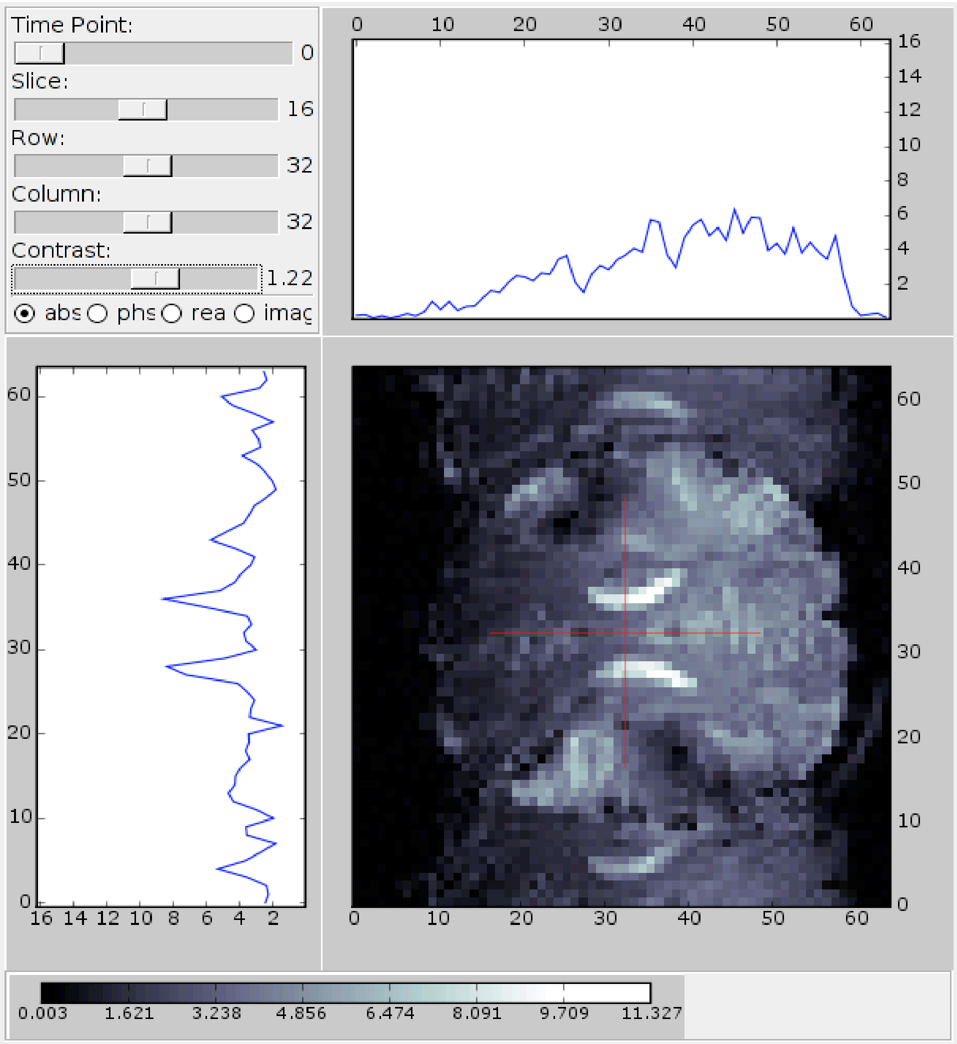


command line options:
input file
output file
file type info
...

```
#!/usr/bin/env python
from imaging.tools.Rerun import Rerun
if __name__=="__main__": Rerun(__file__).run()

## BEGIN OPS LOG

## Read image from file.
#[ReadImage]
## File name prefix for output (extension will be determined by the format).
#filename = /home/mike/bic_sandbox/trunk/testdata/Bal_phs_corr/epidw_one_ref.fid
## File format to write image in.  (Should eventually beauto-detected)
#format = fid
## Volume range over-ride
#vrange = ()

## Reorder image slices from inferior to superior.
#[ReorderSlices]
## Flip slices during reordering.
#flip_slices = False

....
```

executable log file

Nyquist Ghosting



Problem:
k-space scanning is raster-like, so the “extra” phase recorded at sampled points along a column has a non-linear time dependency.

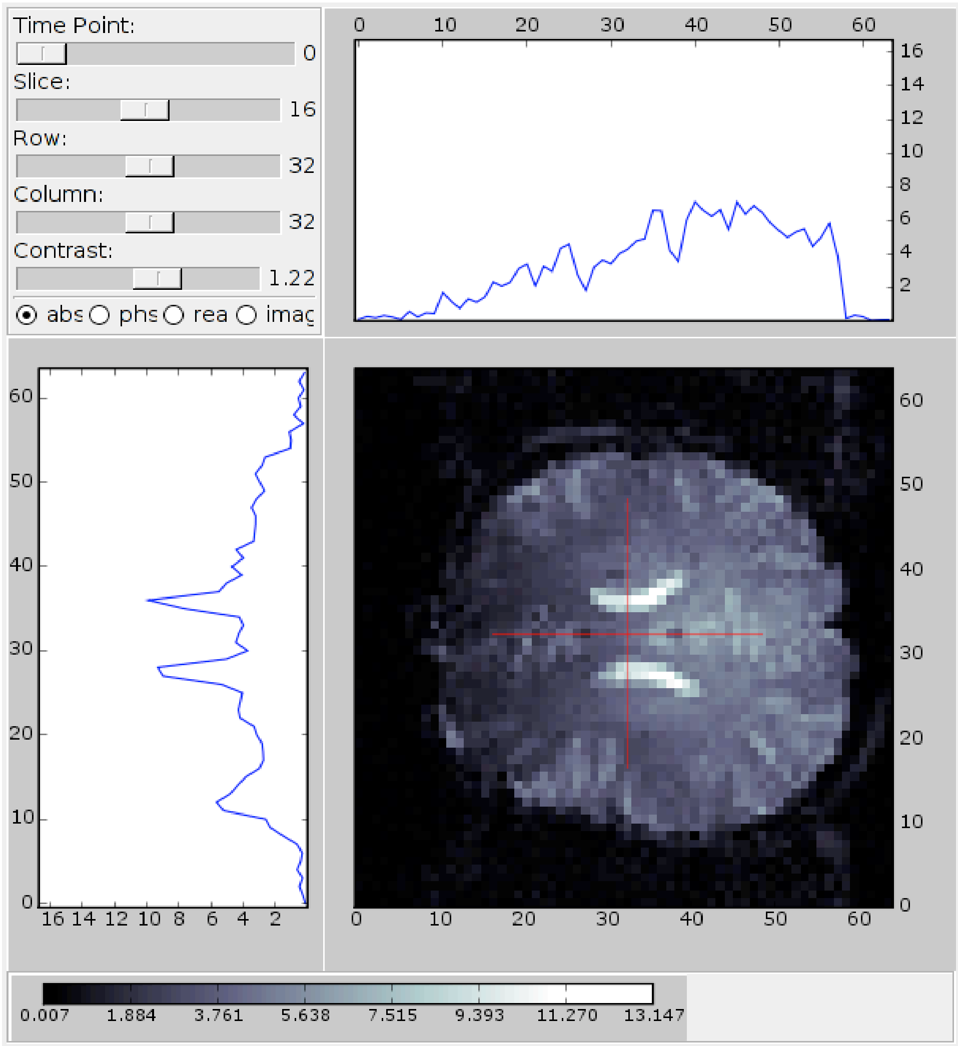
Example of “extra” phase: causal low-pass filter delay

The addition of these introduced phase-nonlinearities to the natural phase gradient in the slow scan direction (k_y in our k-space grid) can cause the total gradient to exceed the sampling limit, and cause ½ FOV ghosting in the image domain.

BAD:
•Loss of SNR
•Interference patterns in region of overlap

Berkeley
University of California

Nyquist Ghost Correction



Solution:
Correct for the phase errors introduced by system delays and gradient offset. A reference scan is used to solve for the model parameters.

A model of phase errors introduced by system:

$$e^{i[\Delta\omega_o+\epsilon_S \mathbf{r}\cdot\delta\mathbf{g}^o][(-1)^\mu \Delta t/2+\mu T_l]}\, e^{i(-1)^\mu g_r^o \mathbf{r}\cdot\delta\mathbf{k}}$$

organize this, and find a relationship to the angle of one Fourier transformed reference echo multiplied by the conjugate of the transform of the following echo:

$$a_1 = \epsilon_S \delta g_r^o \Delta t/2 + g_r^o \delta k_r \qquad a_2 = \epsilon_S \delta g_r^o T_l \qquad a_6 = \Delta\omega_o T_l$$

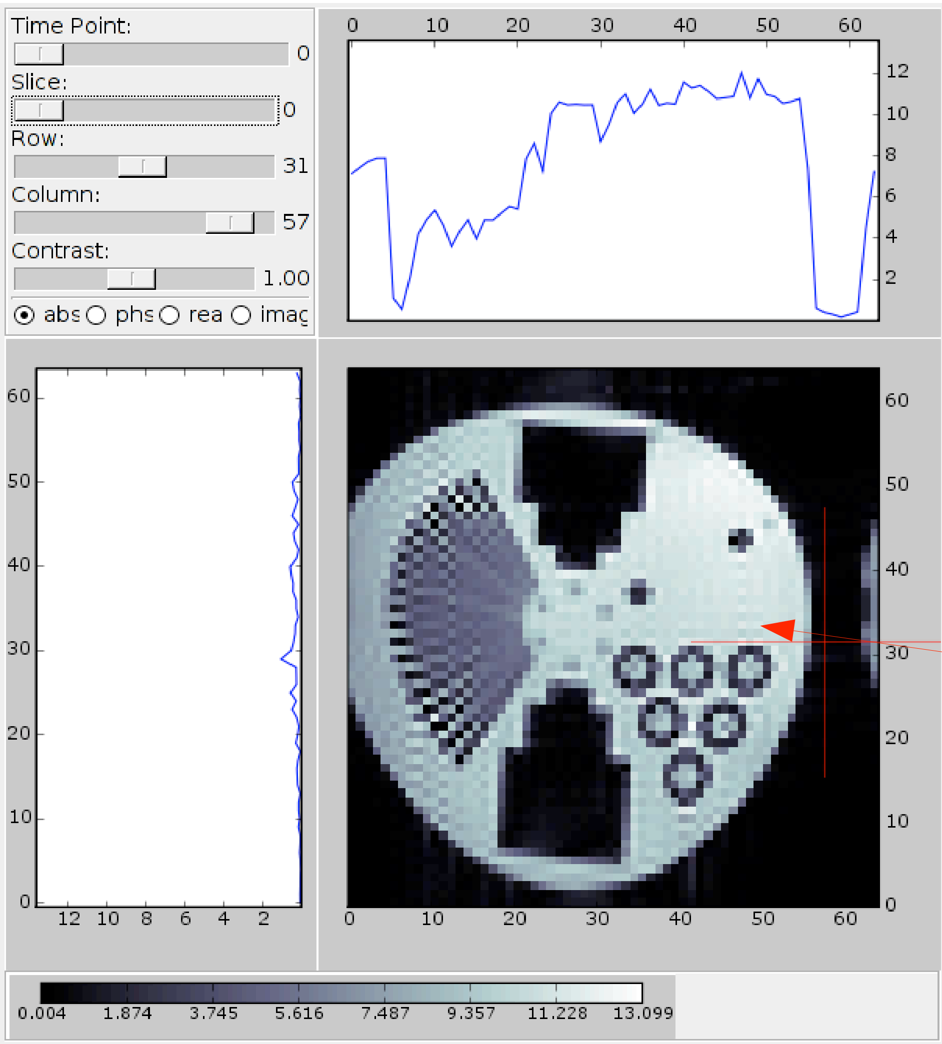
$$a_3 = \epsilon_S \delta g_s^o \Delta t/2 + g_r^o \delta k_s \qquad a_4 = \epsilon_S \delta g_s^o T_l \qquad a_5 = \Delta\omega_o \Delta t/2$$

$$\angle \hat{\mathbb{S}}_s[r;\mu]\hat{\mathbb{S}}_s^*[r;\mu+1] = 2[ra_1+sa_3+a_5](-1)^\mu - ra_2 - sa_4 - a_6$$

solve for these system parameters using SVD and correct the image:

$$S_s[r;m] = e^{-i[ra_1+sa_3+a_5](-1)^m}\, e^{-i[ra_2+sa_4+a_6]m}\, \hat{S}_s[r;m]$$

Geometric Distortion

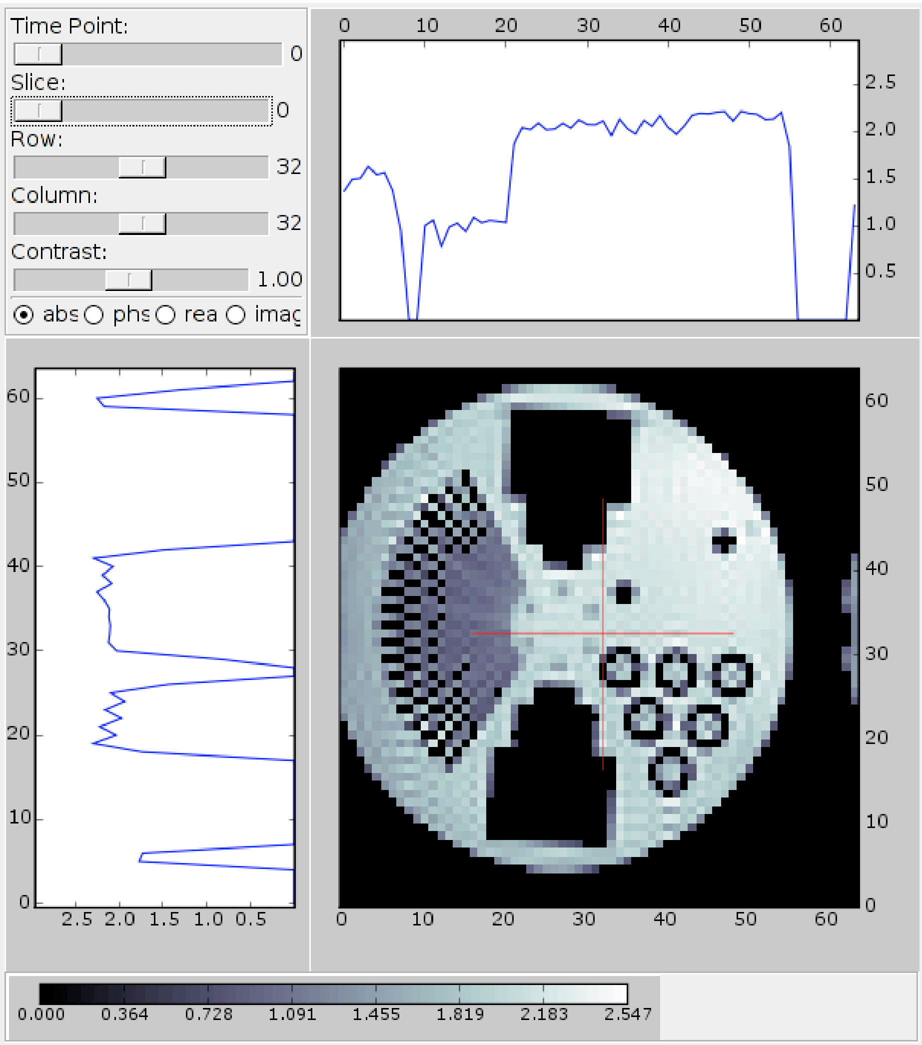


Problem:
Susceptibility-induced field inhomogeneities render the magnetic fields non-linear. Their local effects on magnetization precession distorts the phase to position mapping.

BAD:
•Functional data may not register well with anatomical data

should be round!

Geometric Distortion Correction



Solution:
First calculate a field map from a pair of spin-echo images acquired at different t_0—measures the amount of dephasing due field inhomogeneities.

Knowing the field map and other system values, we can compute the part of the perturbation kernel responsible for this artifact and try to reverse the effect.

$$\mathcal{K}_s[q_1; n_2, n_2'] = \sum_{q_2 \in P} e^{i\pi(n_2' - n_2)q_2/N} e^{in_2\phi_s[q_1,q_2]T_l}.$$

$$S_s[q_1; n_2] = \sum_{n_2'} \mathcal{K}_s^{-1}[q_1; n_2, n_2'] \hat{S}_s[q_1, n_2']$$

CIRL

The END

We are on the WWW at https://cirl.berkeley.edu/view/BIC/ImagingTools

CIRL . BIC . ImagingTools

...collaborate on BIC scientific software.

Search:

Logon | Register | Printable

NAVIGATION

- CIRL
- Helpdesk
- Brain Imaging Center
  - Software
  - Scanning
  - Facilities
- S.F. Bay Area Society for Neuroscience Chapter
- HWNI Graduate Students

Imaging Tools

The imaging-tools package is a software distribution written in Python which provides a number of command-line tools for manipulating MRI data, as well as a code library of reusable components. The following tools comprise the imaging-tools package:

- **recon** : reconstruct k-space MRI scan data and correct for imaging artifacts
- **getparam** : extract parameter values from Varian procpar files
- **dumpheader** : extract header information (main header or block header) from Varian fid files
- **fdf2img** : convert Varian FDF formatted image data into Analyze formatted image data
- **viewimage** : view any reconstructed image with the internal slice-viewer

The code is structured to encourage and enable experimentation and research development. Central object types in the imaging-tools system are images, readers, operations, and writers. Image manipulation tools can be created by composing appropriate combinations of these core objects into a pipeline beginning with a reader, followed by a sequence of operations, and ending with a writer. Image data in the pipeline is represented by an Image object conforming to a conventional Image interface (defined in the system).

New: RPMs are now signed with a CIRL GPG key, which can be downloaded from this page (see the attachments). Admins of RPM-based machines can just do

rpm --import **https://cirl.berkeley.edu/twiki/pub/BIC/ImagingTools/RPM-CIRL-GPG-KEY**

Documentation

- **Installation**
- **User Information**
- **Developer Information**
- **References**

Download

version 0.3 **(latest Beta version)**:

*requires BLAS-optimized Numeric, see release notes or the **depedency page***

- **python-imaging-tools-0.3.tar.gz (source tarball)**
- **python-imaging-tools-0.3-1.i386.rpm (Fedora Core 4 RPM)**
- **Release Notes**