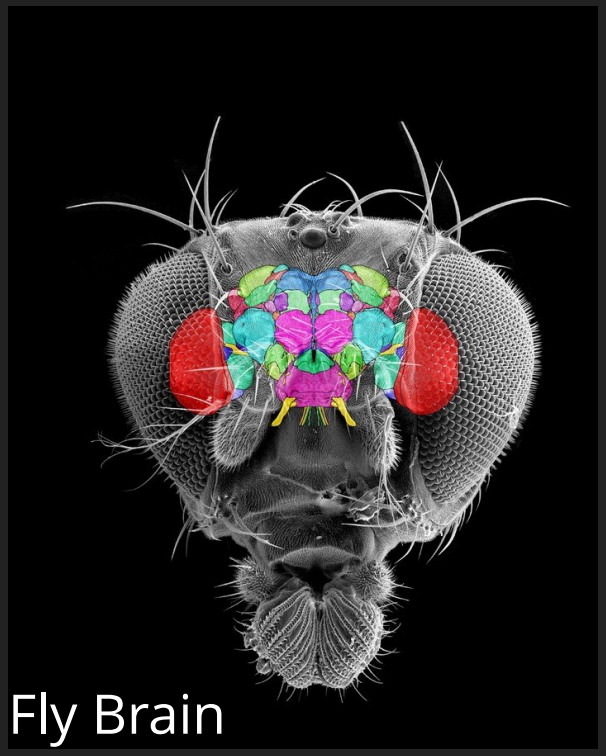
Reverse Engineering Animal Vision with Genetics, Virtual Reality and Python

sdvillal@gmail.com
http://www.strawlab.org

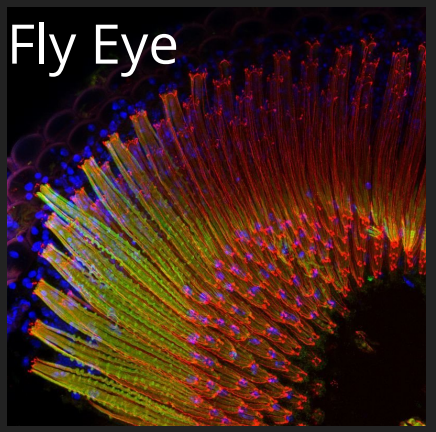
TALK REPOSITORY:

http://github.com/strawlab/strawlab-examples

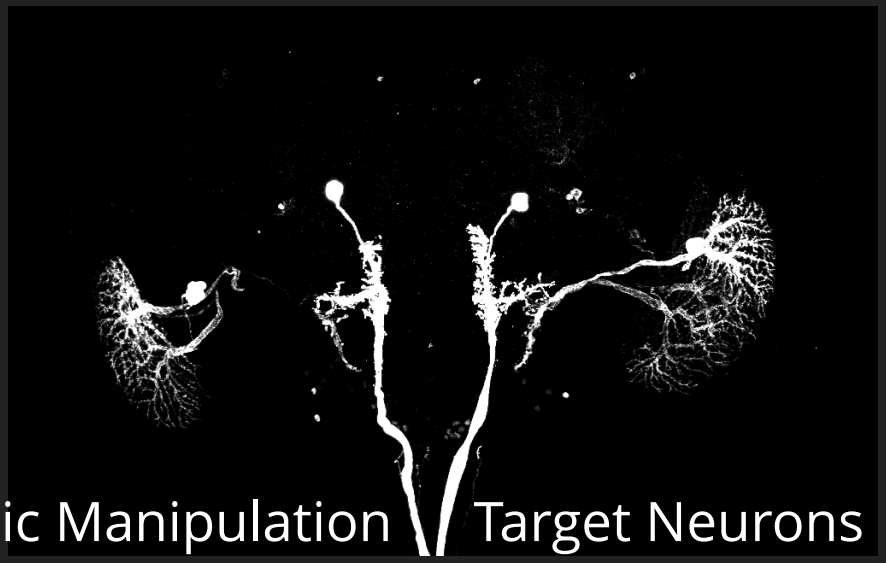
Drosophila Neuroscience



Fly Brain



Fly Eye



Genetic Manipulation \ Target Neurons

Virtual Reality and Tracking

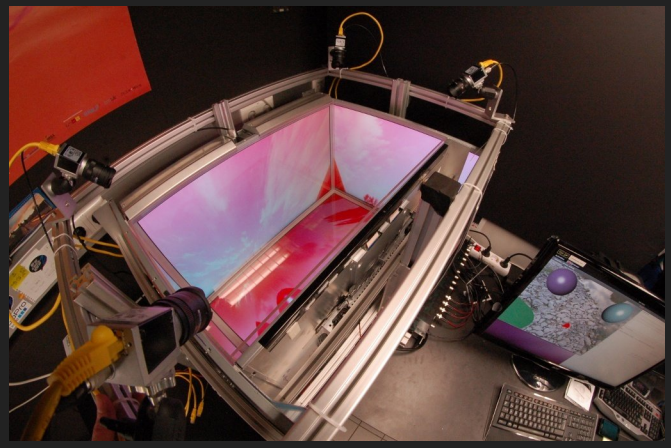
Fish Tracking

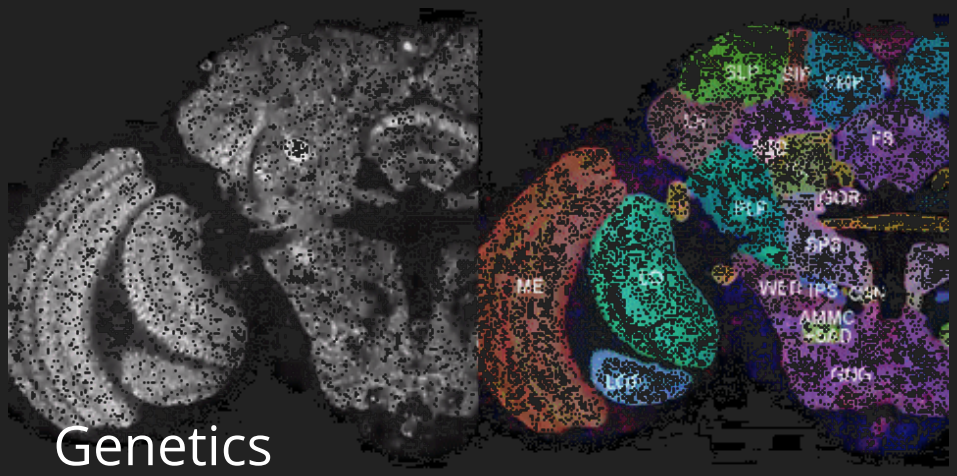
Fooling a Spider

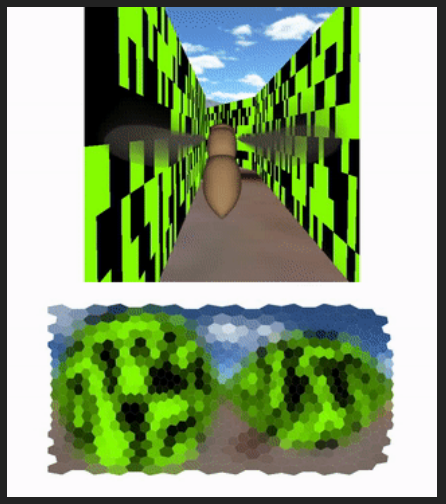
Fooling a GoPro

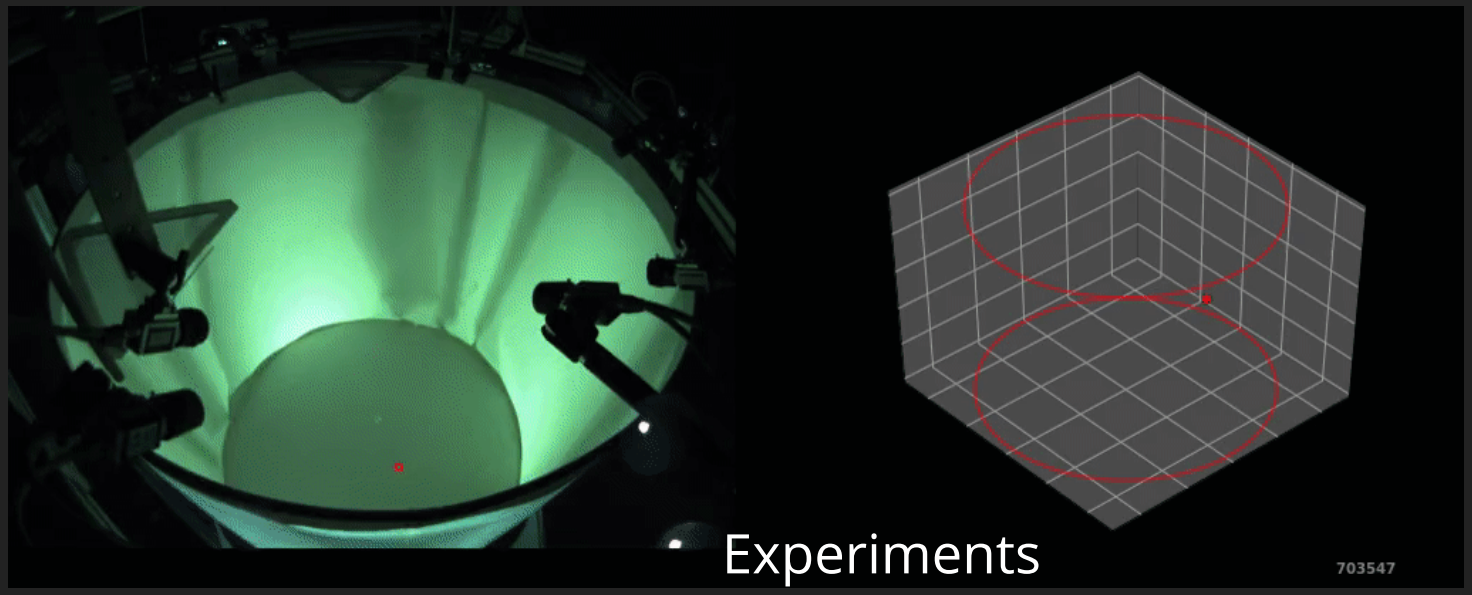
Why?

Behavioural Sampling

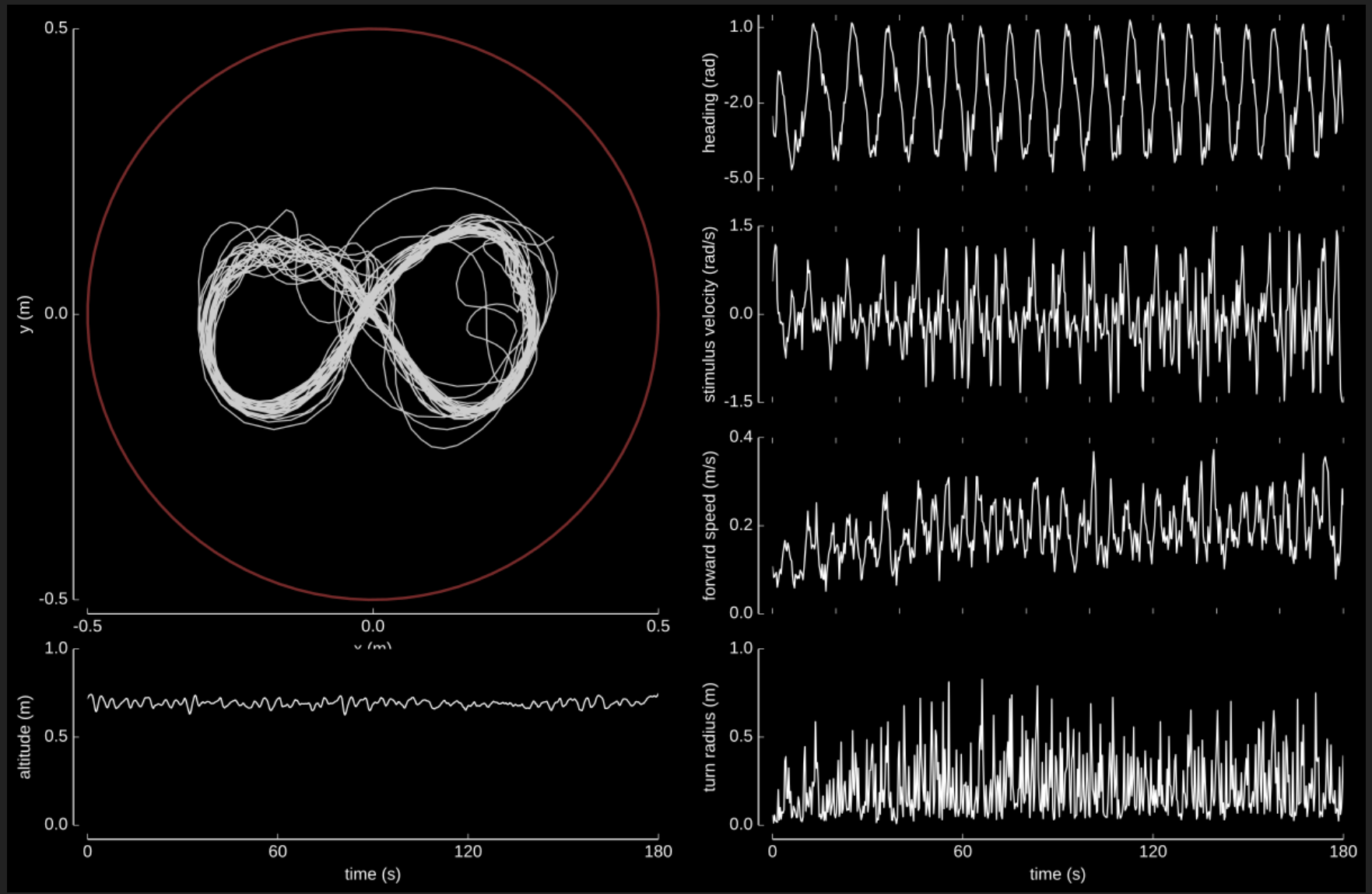



Genetics


Virtual Reality


Experiments
703547

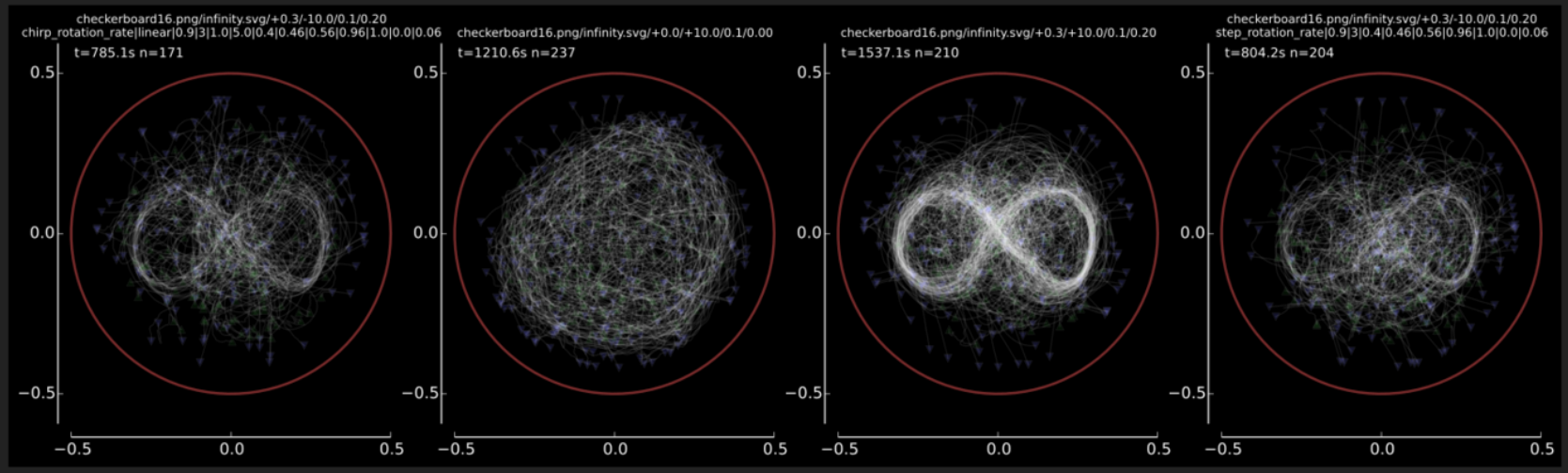
Trials


0.5
0.0
-0.5
y (m)
-0.5
0.0
0.5
x (m)
1.0
0.5
0.0
altitude (m)
0
60
120
180
time (s)
heading (rad)
1.0
-2.0
-5.0
stimulus velocity (rad/s)
1.5
0.0
-1.5
forward speed (m/s)
0.4
0.2
0.0
turn radius (m)
1.0
0.5
0.0
0
60
120
180
time (s)

Largish Datasets

A typical dataset of today has...

- over 300 experiments (overnight)
- over 400K trials, 300M observations
- over 15G in single precision (might become 1.5T soon)



We have

Biologists using virtual machines on laptops, slow wireless networks, data too large to load on memory or fast.

We want

A store that supports loading the time series from a very concrete type of queries fast and easy on memory *

```
tseries.query('genotype="superfly" and length_s > 2')
```

* Pickles can get the job done quite well in batch workloads

Many Good Offers

Columnar

Chunked

Compressed

Blocked

Preconditioned

Jagged

Get Your Time Series Faster

http://github.com/sdvillal/jagged

The simplest API

```
from jagged.blosc_backend import JaggedByBlosc

# Instantiate
jagged = JaggedByBlosc(path='/tmp/jagged-by-blosc')

# Append
index1 = jagged.append(x)
index2 = jagged.append(y)
index3 = jagged.append(y)

# Read
arrays = jagged.get([index1, index2, index3])
```

A backend for each child

Backend	comp	chunk	column	mmap	lin	lazy	cont
JaggedByBlosc	X			X			
JaggedByCarray	X	X			X		X
JaggedByH5Py	X	X			X	X	X
JaggedByJoblib	X	X					
JaggedByMemMap				X	X	X	X
JaggedByNPY							
JaggedByBloscpack	X						
JaggedByPickle	X	X					

It works

http://github.com/sdvillal/jagged

15GB to 11GB at no considerable speed loss
or even speed gain for usual queries*

* Thanks to blosc preconditioning and lz4hc compression
Pickles are a tough baseline
Find the benchmark code in the talk repository
https://github.com/strawlab/strawlab-examples

Lesson Learned: Benchmark

- Benchmark compressibility first

Our data is not compressible without preconditioning

```
# Compress this...
x = np.linspace(0, 10000, 10000, dtype=np.float32)

# Without shuffling: cr=0.99 (same size)
# With shuffling: cr: 5.02 (5 times smaller)
```

Our data gets better compressed if column-wise

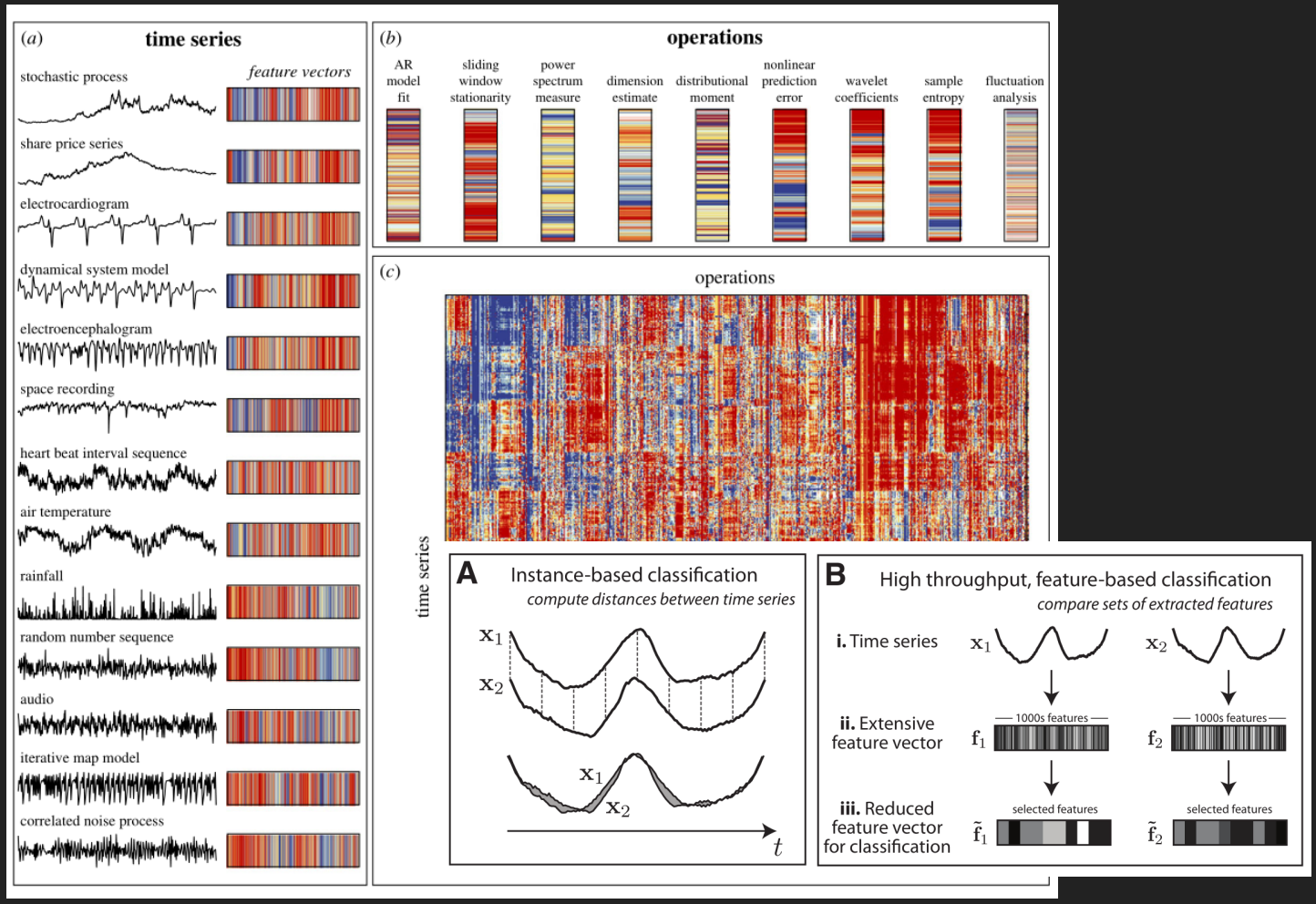
Our data does not like when compressed in chunks

- Benchmark using real queries

How to analyse?

Just extract features

Highly Comparative Time Series Analysis (HCTSA)



Fulcher, Little and Jones, 2013

But it is Matlab

Pyopy

Talk to Matlab (People)

http://github.com/strawlab/pyopy

Pyopy two goals

Help creating pythonic bindings to matlab libraries that work in octave too

Provide bindings for the HCTSA time-series feature extraction library*

* Will be soon fully open-sourced

Pyopy: 100+ classes

```
class EN_PermEn(HCTSASuper):
    """
    Matlab doc:
    ----------------------------------------
    % EN_PermEn     Permutation Entropy of a time series.
    %
    % "Permutation Entropy: A Natural Complexity Measure for Time Series"
    % C. Bandt and B. Pompe, Phys. Rev. Lett. 88(17) 174102 (2002)
    %
    %---INPUTS:
    % y, the input time series
    % m, the embedding dimension (or order of the permutation entropy)
    % tau, the time-delay for the embedding
    ...
    """

    def __init__(self, m=3.0, tau='ac'):
        super(EN_PermEn, self).__init__()
        self.m = m
        self.tau = tau
```

Pyopy: example

http://github.com/strawlab/pyopy

```
from pyopy.hctsa import hctsa
from whatami import what2id

hctsa.prepare()

extractors = [hctsa.operations.EN_PermEn_4_1,
              hctsa.bindings.EN_PermEn(m=6)]

for extractor in extractors:
    for x in time_series:
        features = extractor.compute(x)
        print(what2id(extractor), features)
```

Whatami

Identify your computations

http://github.com/sdvillal/whatami

```
@whatable
class FeatureExtractor(object):

    def compute(self, x):
        """Returns a tuple with features values."""
        raise NotImplementedError()

    def _outnames(self):
        return ()

    def _infonames(self):
        return ()

    def fnames(self):
        if not self._outnames():
            outnames = [what2id(self)]
        else:
            outnames = whatadd(self.what(),
                               'out', self._outnames())
        return outnames + whatadd(self.what(),
                                  'info', self._infonames())
```

```
class PermEn(FeatureExtractor):
    """
    Computes Bandt and Pompe permutation entropy.

    Parameters
    ----------
    column : string, default dtheta
      The time series columns

    order : int [2...], default 2
      The embedding dimension
      (= length of alphabet symbols)

    normalize : boolean, default True
      Make the result to be in [0, 1]

    return_counts : boolean, default False
      Return the symbol distribution too?
    """
```

```
>>> from whatami import what2id, id2what

>>> permen3 = PermEn(column='speed',
...                  order=3,
...                  return_counts=True))

>>> print(what2id(permen3))
PermEn(column='speed',normalize=False,order=3)
# Note:
#  - The parameters have been sorted by name
#  - "return_counts" is not part of the id string
#    (since it does not change the computation result)

>>> print('\n'.join(permen3.fnames()))
PermEn(column='speed',normalize=False,order=3)
PermEn(column='speed',info='counts',normalize=False...
# Note: these are the names of our features
# We can manipulate them easily with whatami's parser
# We use dicts to map them to human-friendly strings
```

Whatami goodies

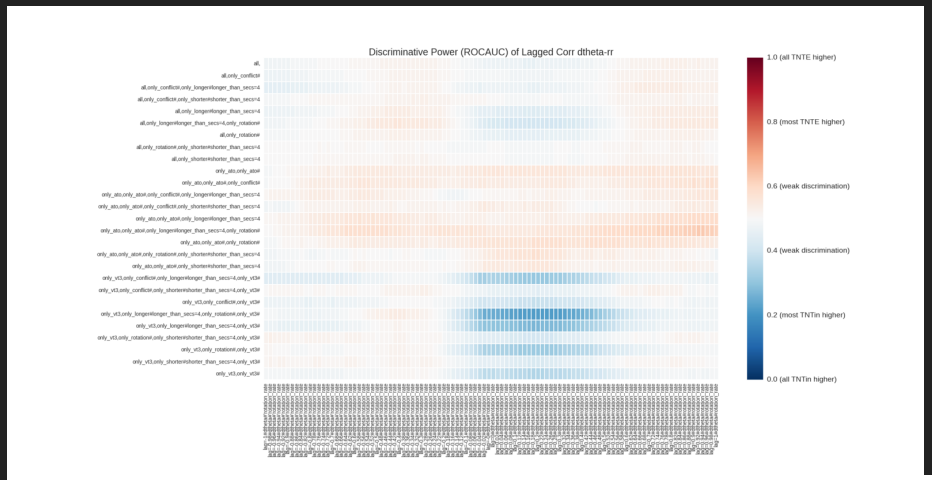
http://github.com/sdvillal/whatami

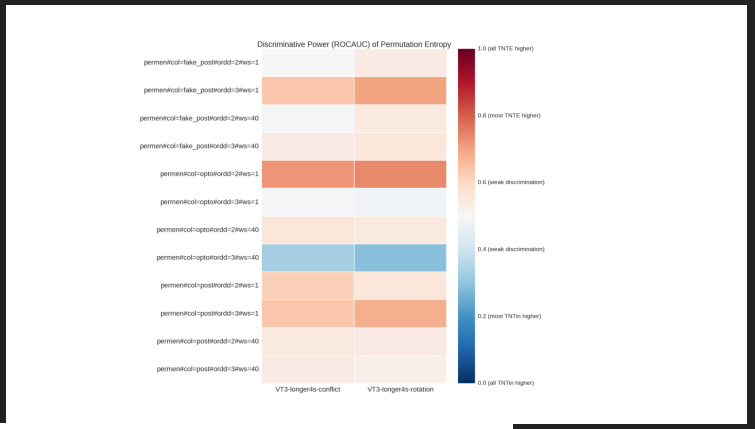
- Dead simple, unobtrusive, imposes nothing.
- Introspection deals with many types (including functions, partials, curries and even closures, plus numpy arrays and pandas objects).
- Introspection deals with __slots__, __dict__, descriptors and closures.
- Introspection deals with nested structures.
- Plugins to add custom types.
- Bindings to libraries (e.g. scikit-learn)
- ID string parsing.
- Helpers for data tidying (e.g. melting).

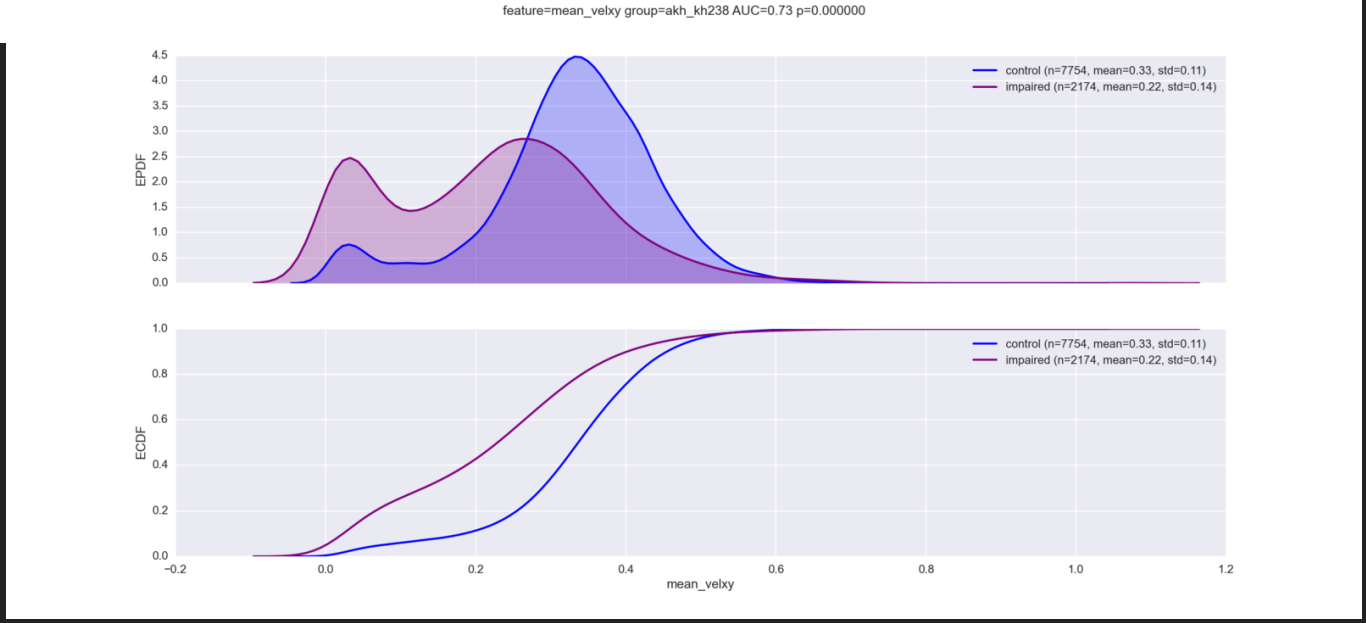
Example Analysis

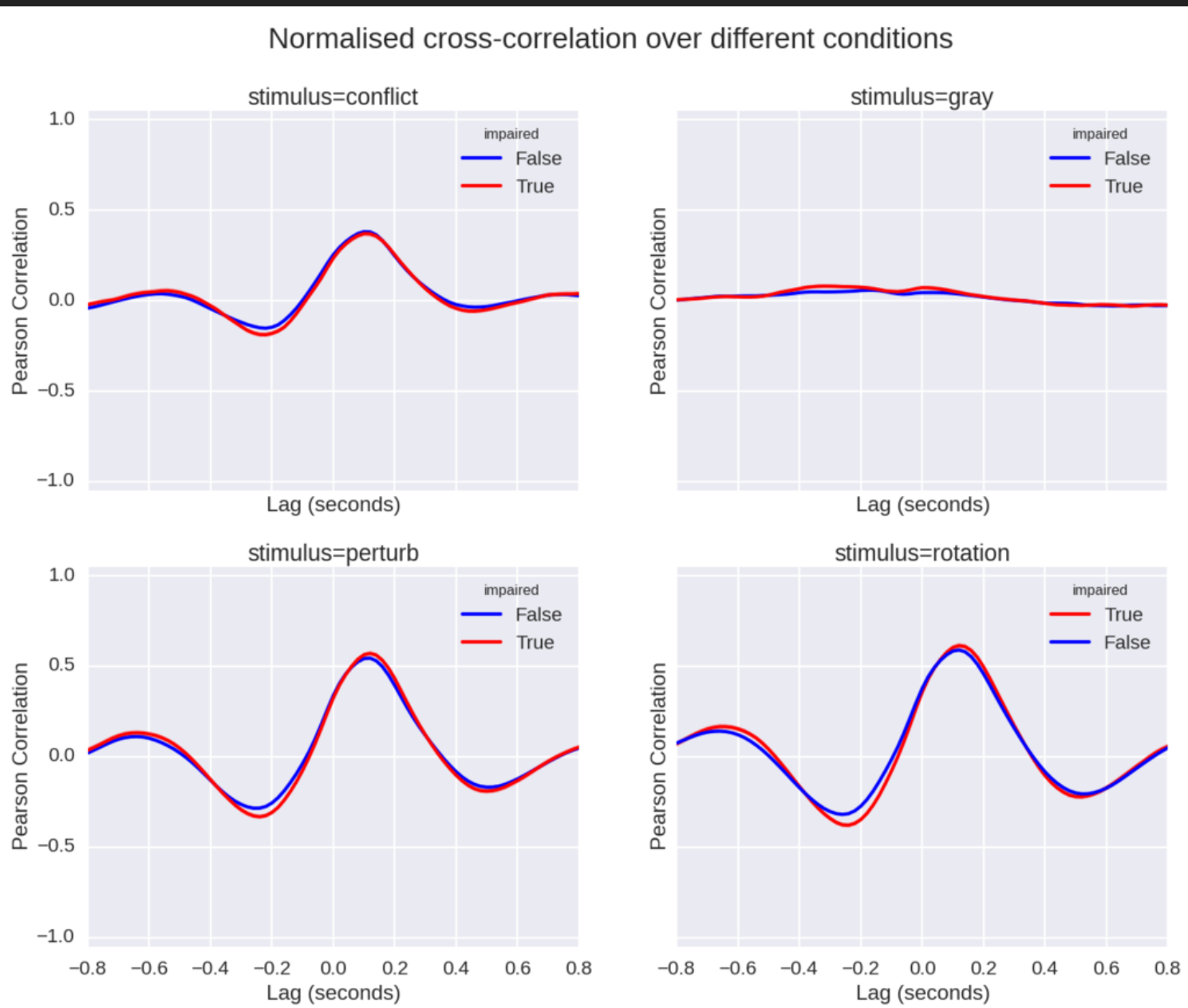
Data Included!

http://github.com/strawlab/strawlab-examples


Discriminative Power (ROCAUC) of Lagged Corr dtheta-rr
1.0 (all TNTE higher)
0.8 (most TNTE higher)
0.6 (weak discrimination)
0.4 (weak discrimination)
0.2 (most TNTin higher)
0.0 (all TNTin higher)


Discriminative Power (ROCAUC) of Permutation Entropy
permen#col=fake_post#ordd=2#ws=1
permen#col=fake_post#ordd=3#ws=1
permen#col=fake_post#ordd=2#ws=40
permen#col=fake_post#ordd=3#ws=40
permen#col=opto#ordd=2#ws=1
permen#col=opto#ordd=3#ws=1
permen#col=opto#ordd=2#ws=40
permen#col=opto#ordd=3#ws=40
permen#col=post#ordd=2#ws=1
permen#col=post#ordd=3#ws=1
permen#col=post#ordd=2#ws=40
permen#col=post#ordd=3#ws=40
VT3-longer4s-conflict
VT3-longer4s-rotation
1.0 (all TNTE higher)
0.8 (most TNTE higher)
0.6 (weak discrimination)
0.4 (weak discrimination)
0.2 (most TNTin higher)
0.0 (all TNTin higher)

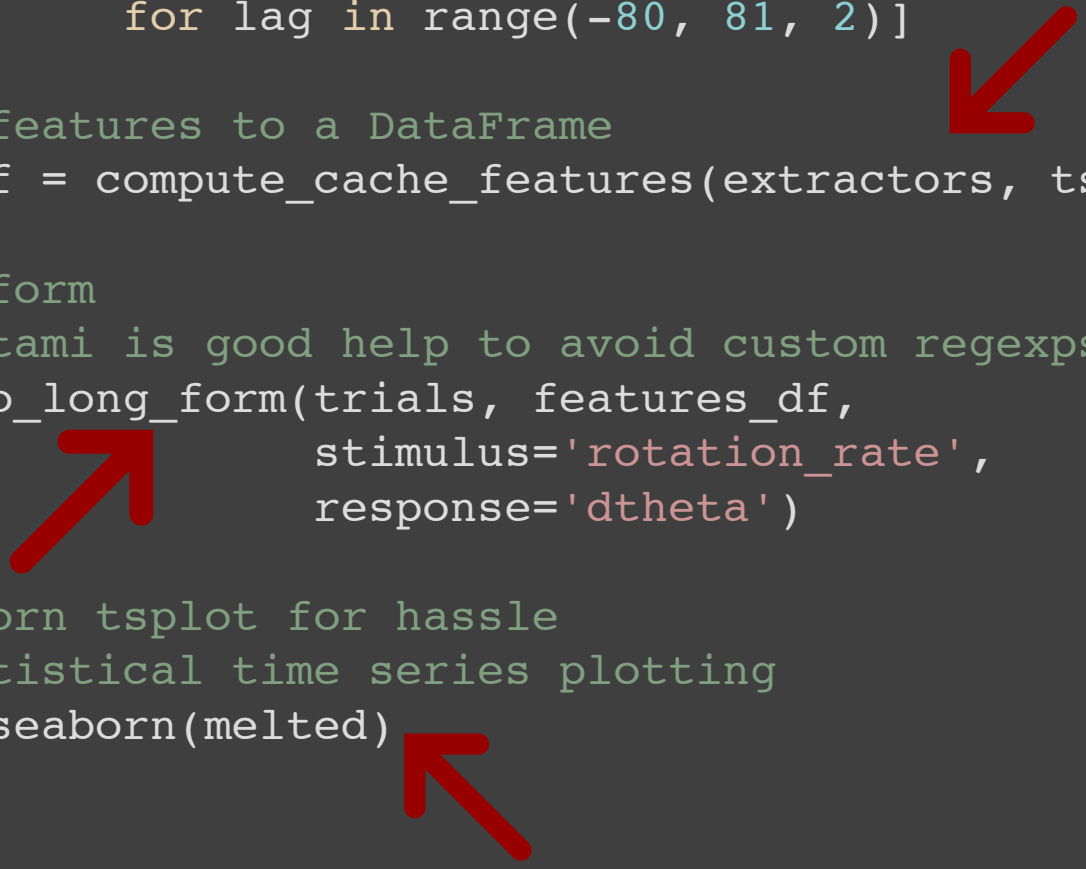

feature=mean_velxy group=akh_kh238 AUC=0.73 p=0.000000
EPDF
control (n=7754, mean=0.33, std=0.11)
impaired (n=2174, mean=0.22, std=0.14)
ECDF
control (n=7754, mean=0.33, std=0.11)
impaired (n=2174, mean=0.22, std=0.14)
mean_velxy


Normalised cross-correlation over different conditions
stimulus=conflict
stimulus=gray
stimulus=perturb
stimulus=rotation
impaired
False
True
Pearson Correlation
Lag (seconds)
1.0
0.5
0.0
−0.5
−1.0
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8

```
DATA_PATH = ...

# We will work with a smallish dataset:
#   - 51359 trials
#   - 18328689 (18M) observations
#   - 1.7GB (uncompressed, single precision)
# You can download it from:
#   https://zenodo.org/record/29193

# Download released dataset
download_degradation_dataset(dest=DATA_PATH)

# Data hub, access anything in the data
hub = FreeflightHub(path=DATA_PATH)

# Put the time series in a jagged store
# This jagged instance allows "lazy" pandas DataFrames
jagged = JaggedByMemMap(path=op.join(DATA_PATH, 'jagged-mmap'),
                        autoviews=True, contiguity='auto')
hub2jagged(hub, jagged)
```

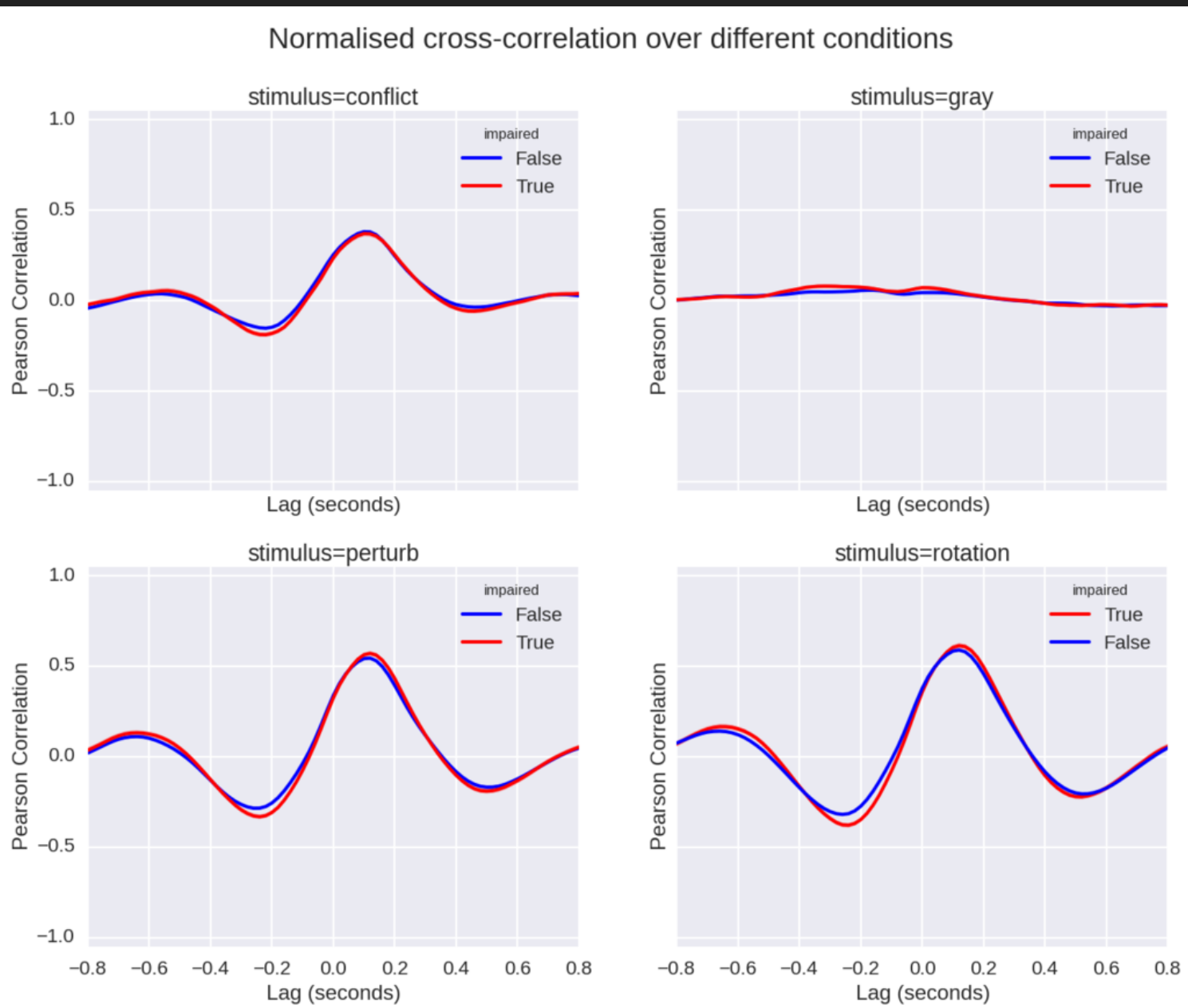
```
# The trials DataFrame
trials_df = hub.trials_df()

# Time-series names
columns, human_friendly = hub.columns(as_pandas_index=True)

# Fast retrieval of time-series for concrete trials
pandify = partial(pandify, columns=human_friendly)
get_tseries = partial(jagged.get, factory=pandify)


# Retrieve a subset of the trials
trials = trials_df.query('length_s > 2 and '
                         'exp_group == "DPPIII"').reset_index()
print('There are %d trials' % len(trials))

# Retrieve their time series
tseries = list(get_tseries(trials.jagged_index))
```

```
# Feature extractors
extractors = [PermEn(order=3),
              PermEn(order=3, column='speed_xy')]
extractors += [LaggedPearson(lag=lag,
                             stimulus='rotation_rate',
                             response='dtheta')
               for lag in range(-80, 81, 2)]

# Extract features to a DataFrame
features_df = compute_cache_features(extractors, tseries)

# To long form
# Here whatami is good help to avoid custom regexps
melted = to_long_form(trials, features_df,
                      stimulus='rotation_rate',
                      response='dtheta')

# Use seaborn tsplot for hassle
# free statistical time series plotting
plot_with_seaborn(melted)
```



```
# This is how whatami helps us
def to_long_form(stimulus, response):
    ...
    # Lets just get the lagged correlation we want
    lag_columns = whatselect(fnames,
                             name='LaggedPearson',
                             kvs=[('stimulus', stimulus),
                                  ('response', response)])
    melted = melted[melted['fname'].isin(lag_columns)]
    # Extract the lag to a new column
    melted = whatid2columns(melted, 'fname',
                            columns=['lag'], inplace=False)
    melted['lag'] *= melted['dt']
    ...
```

```
# This is how plot lagged looks like
def plot_lagged(melted, ax, title):
    sns.set_context('poster')
    sns.tsplot(melted,
               time='lag',
               unit='unit',
               condition='impaired',
               value='value',
               estimator=np.nanmean,
               ax=ax,
               color={True: 'r', False: 'b'})
    ax.set_title(title)
    ax.set_ylabel('Pearson Correlation')
    ax.set_xlabel('Lag (seconds)')
    ax.set_ylim((-1.05, 1.05))

# A generic plot function for cross-correlations
```


Normalised cross-correlation over different conditions
stimulus=conflict
stimulus=gray
stimulus=perturb
stimulus=rotation
impaired
False
True
Pearson Correlation
Lag (seconds)
1.0
0.5
0.0
−0.5
−1.0
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8

Acknowledgements

- Stained brain and fly eye by Karin Panser
- Virtual Reality videos and other plots by John Stowers
- Fish tracking video by Max Hofbauer
- Model of retinal input to the fly by Andrew Straw
- Fly Brain Art and Brain Atlas video from Ito et. al., 2013
- HCTSA figures from Fulcher et. al. 2013