Rapid Development with the BCILAB, SNAP and LSL Platforms

Christian A. Kothe

SCCN, INC, UCSD

Outline

1. Overall Experimentation Environment
2. The Lab Streaming Layer (LSL)
3. Simulation and Neuroscience Application Platform (SNAP)
4. The BCILAB Toolbox
    1. Toolbox Overview
    2. Workflows and Concepts
    3. In-Depth Walkthrough
    4. Adding New Methods
5. Further Reading

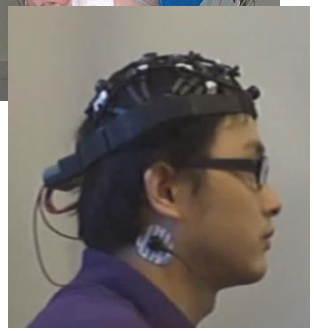
Swartz Center for Computational Neuroscience

1 Overall Experimentation Environment

Goals

- Enable experiments involving acquisition of multi-modal brain- and bio-signals from a variety of sources, such as:

EEG and ExG	Full-Body Motion Capture	Eye-Tracking	Human Interface Devices, System State

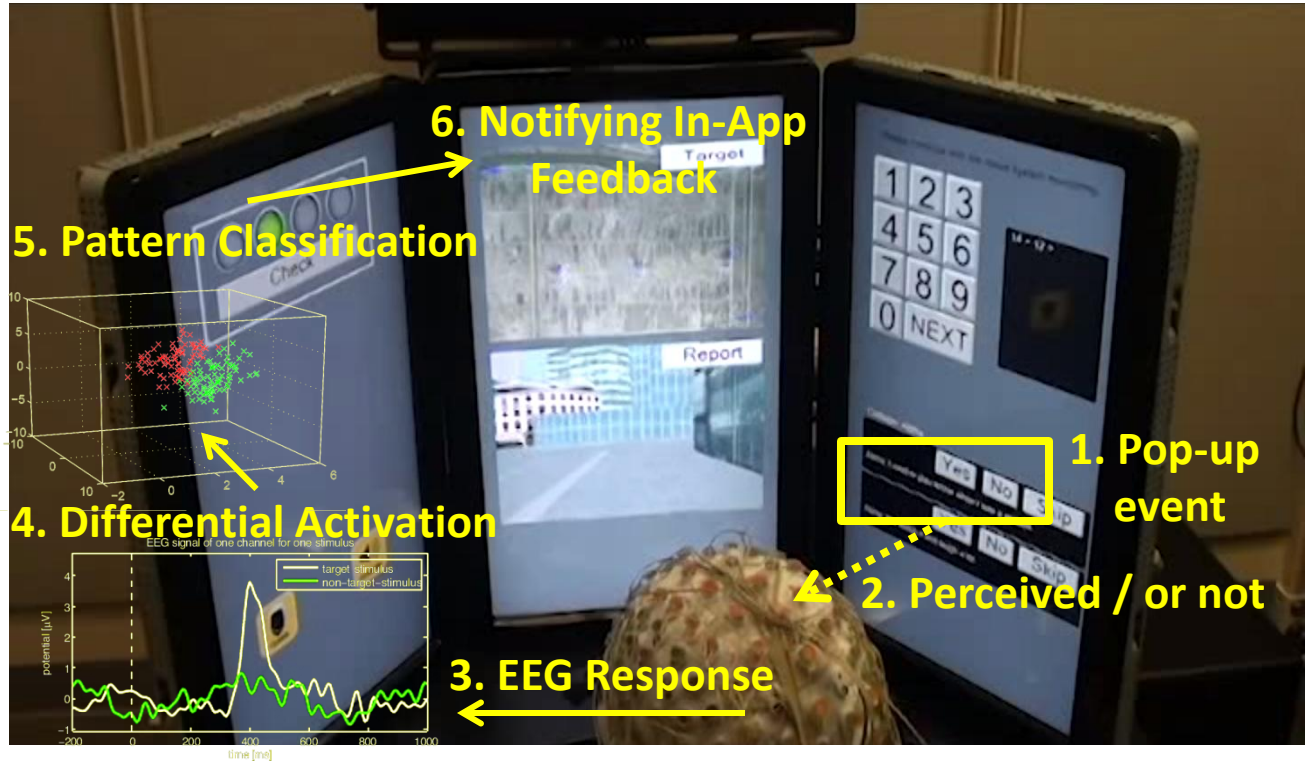


Goals

- Enable experiments involving complex scripting and multi-subject interactions

Goals

- Enable experiments depending on advanced real-time analysis of acquired data



Disclaimer: Not tested yet for this experiment!

Goals

- Enable unhindered offline analysis of the data:
  - Retain a complete record of experiment events, meta-data
  - All measures time-synchronized
  - Well-organized file format
  - Wide range of compatible analysis tools: EEGLAB, MoBILAB, BCILAB, SIFT, …

Real-Time Components Overview

- Lab Streaming Layer (LSL)
  - Underlying distributed data acquisition, transport and collection system
- Simulation and Neuroscience Application Platform (SNAP)
  - Scalable experiment scripting environment based on Python and Panda3d
- Brain-Computer Interface Laboratory (BCILAB)
  - Design, prototyping and testing environment for brain-computer interfaces and other cognitive monitoring tools

Swartz Center for Computational Neuroscience

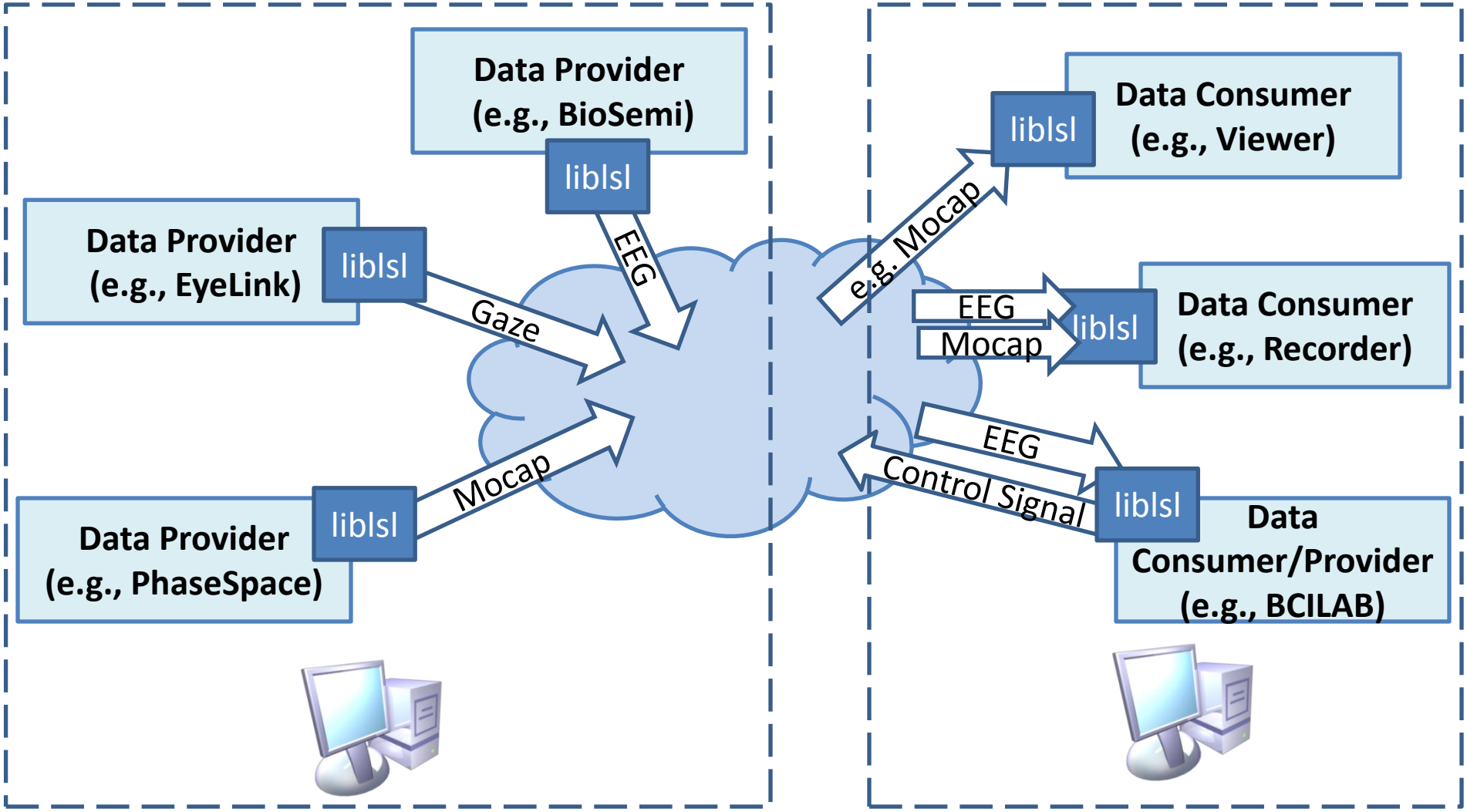
2 The Lab Streaming Layer

code.google.com/p/labstreaminglayer

Purpose

- Provide a *simple and unified* way to distribute and access *experiment time series and meta-data* from all acquisition devices (and other sources)
- Both in *real time and for offline* processing
- Handle *networking*, *time synchronization*, and *fault tolerance* transparently for most client applications
- Support a wide range of hardware out of the box (‘batteries included’)

Overall Layout

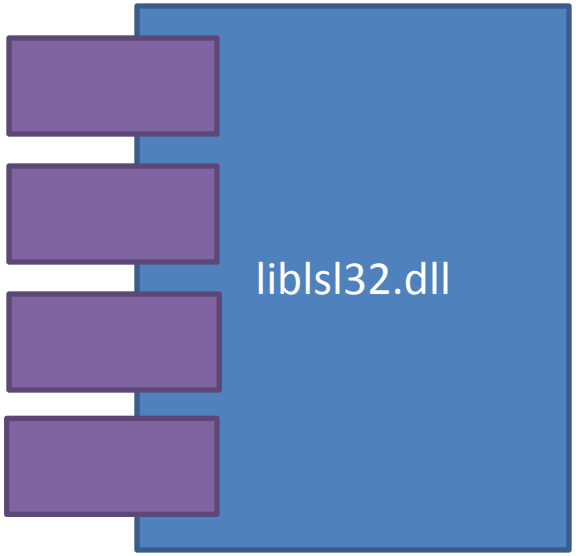


LSL Core Library

- Cross-platform library (MacOS/Win/Linux, 32/64), open source (MIT license)
- Stable interfaces for C, C++, MATLAB, Python with identical feature set
- Robust and clean implementation, stress-tested for days
- Extensively documented, incl. 10s of examples programs
- High throughput (>50KHz) and low latency (<1ms); designed to scale up to large experiments

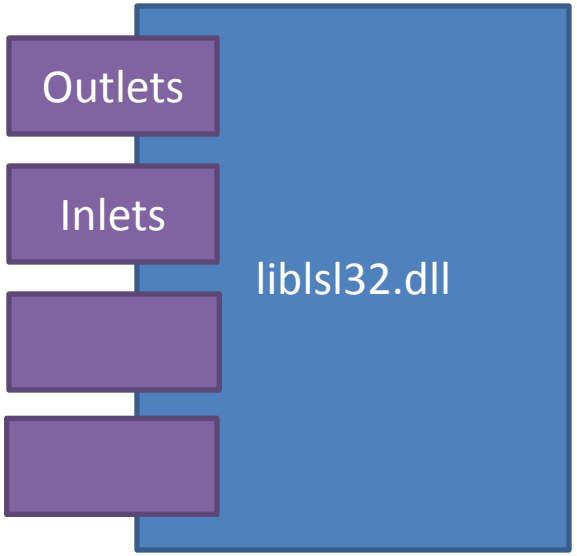
LSL API

- Applications interact with LSL as “producers” or “consumers”



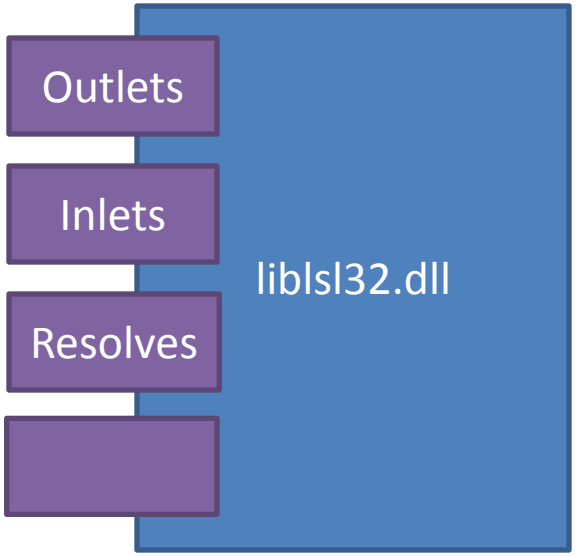
LSL API

- Applications interact with LSL as “producers” or “consumers”
- Producers create one or more *outlets* and push samples in
- Consumers create one or more *inlets* and pull samples out



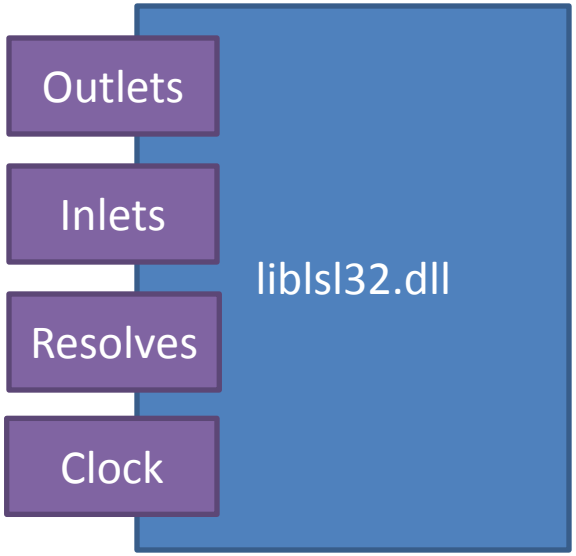
LSL API

- Applications interact with LSL as “producers” or “consumers”
- Producers create one or more *outlets* and push samples in
- Consumers create one or more *inlets* and pull samples out
- Consumers can *resolve* existing data streams on the network (e.g., by name or type)



LSL API

- Applications interact with LSL as “producers” or “consumers”
- Producers create one or more *outlets* and push samples in
- Consumers create one or more *inlets* and pull samples out



- Consumers can *resolve* existing data streams on the network (e.g., by name or type)
- All can use a built-in synchronized *clock*

Examples Programs in MATLAB

```
% instantiate the library
lib = lsl_loadlib();

% make a new stream outlet (name: BioSemi, type: EEG, 8 channels, 100Hz)
info = lsl_streaminfo(lib,'BioSemi','EEG',8,100,'cf_float32','myuid');
outlet = lsl_outlet(info);

% send data into the outlet, sample by sample (8 random numbers each)
while true
    outlet.push_sample(randn(8,1));
    pause(0.01);
end
```

Data Provider

```
% instantiate the library
lib = lsl_loadlib();

% try resolve an EEG stream...
result = {};
while isempty(result)
    result = lsl_resolve_byprop(lib,'type','EEG'); end

% create a new inlet from the first result
inlet = lsl_inlet(result{1});

while true
    % get data from the inlet and print it
    [vec,ts] = inlet.pull_sample();
    fprintf('%.2f\t',vec); fprintf('%.5f\n',ts);
end
```

Data Consumer

Other Languages

```cpp
#include "../../../include/lsl_cpp.h"
#include <stdlib.h>
using namespace lsl;

/**
 * This is an example of how a simple data stream can be offered on t
 * The transmitted samples contain random numbers (and the sampling r
samples).
 */

int main(int argc, char* argv[]) {

	// make a new stream_info (128ch) and open an outlet with it
	stream_info info("SimpleStream","EEG",128);
	stream_outlet outlet(info);

	// send data forever
	float sample[128];
	while(true) {
		// generate random data
		for (int c=0;c<128;c++)
			sample[c] = (rand()%1500)/500.0-1.5;
		// send it
		outlet.push_sample(sample);
	}

	return 0;
}
```

C++

Python

```python
import sys; sys.path.append('..')  # make sure that pylsl is found (note: in a
import pylsl
import random
import time

# first create a new stream info (here we set the name to BioSemi, the content-
# The last value would be the serial number of the device or some other more or
recover).
info = pylsl.stream_info('BioSemi','EEG',8,100,pylsl.cf_float32,'dsffwerwer');

# next make an outlet
outlet = pylsl.stream_outlet(info)

print("now sending data...")
while True:
	# make a new random 8-channel sample; this is converted into a pylsl.ve
	mysample = pylsl.vectorf([random.random(),random.random(),random.random
	# now send it and wait for a bit
	outlet.push_sample(mysample)
	time.sleep(0.01)
```

Misc API Features

- Attaching/receiving XML meta-data

```
for (int k=0;k<8;k++)
    info.desc().append_child("channel")
    .append_child_value("name",channels[k])
    .append_child_value("unit","microvolts")
    .append_child_value("type","EEG");
```

- Handling data at chunk granularity

```
outlet.push_chunk(randn(8,50));

[chunk,stamps] = inlet.pull_chunk();
```

- Handling string-formatted streams (Events, …)

```
outlet.push_sample({'test','123'});
```

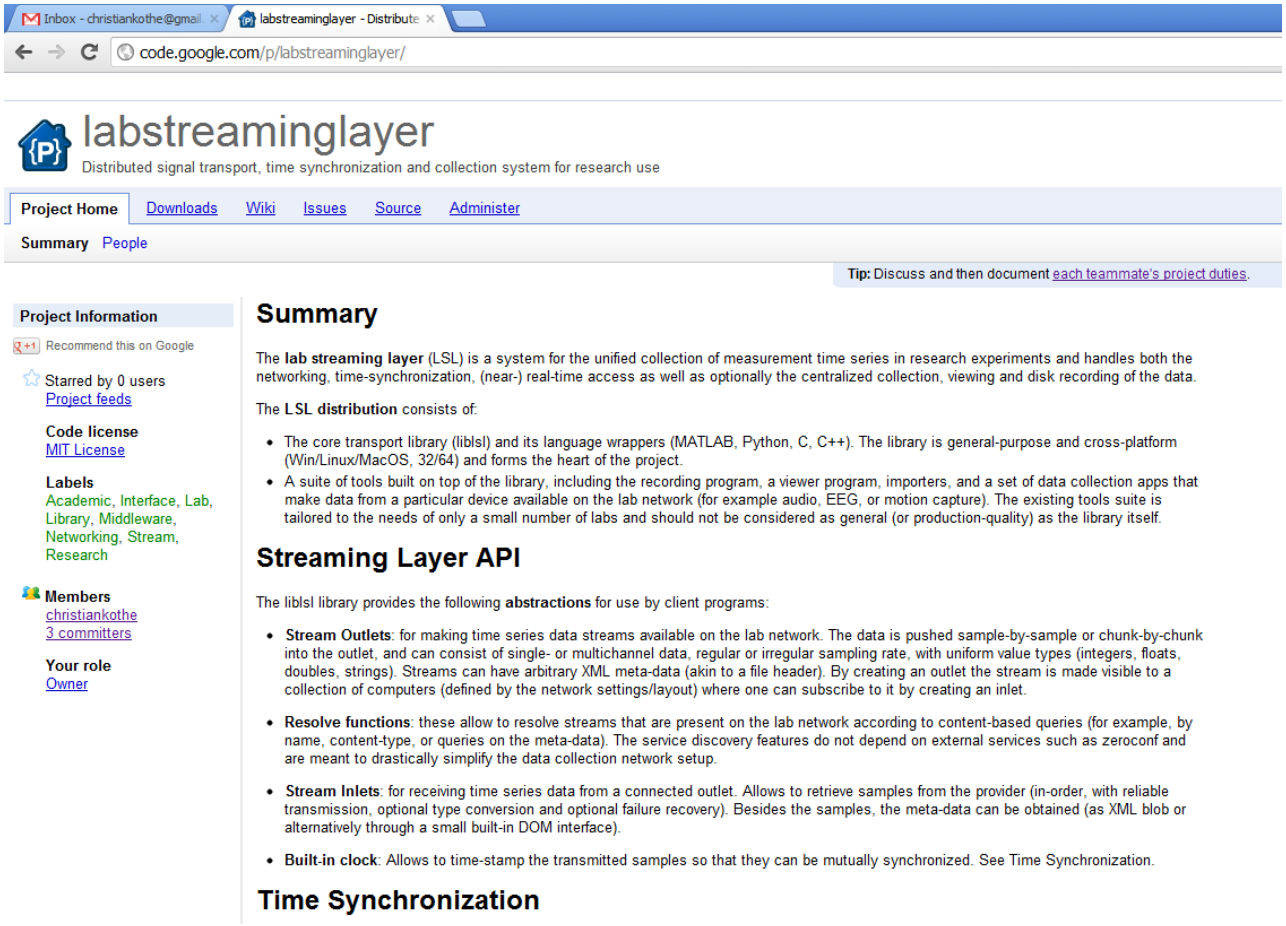
Time Synchronization

- Uses a protocol similar to NTP, achieves sub-ms accuracy on a local network
- Time synchronization only applies to the computers' clocks
- Any uncertainty in when a sample was measured (e.g., due to hardware buffers) remains and cannot be fixed by the library
- In these cases trigger & sync cables can help…

Advanced Network Configuration

- Visibility between LSL applications is restricted to a *scope* that can be set (in a .cfg file) to:
  - Local machine
  - Local router or VPN (**default**)
  - Local “site”
  - Local “organization”
  - Global
  - All clients that have a given group name
  - A set of IPs/Hostnames
- Firewall restrictions (!) and router restrictions apply

LSL Distribution



Available at code.google.com/p/labstreaminglayer

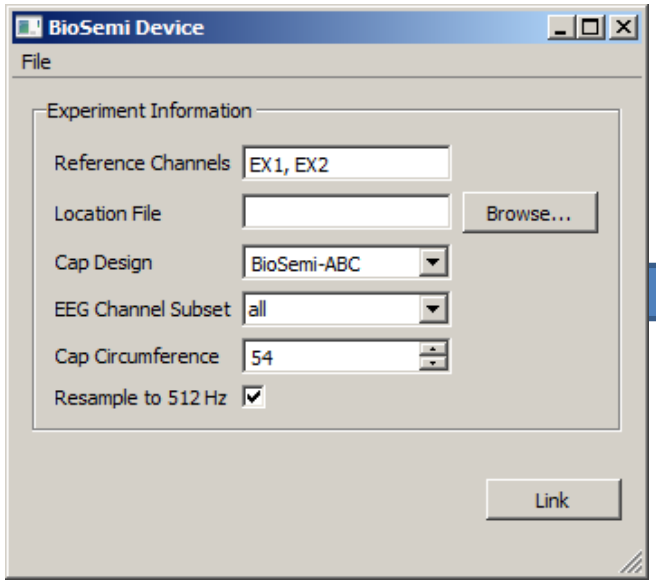
Included with the Distribution

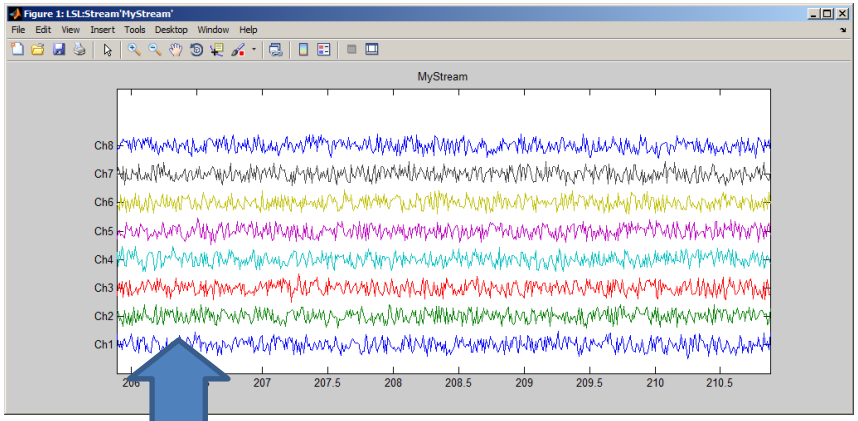
- **Core library**, source, documentation, samples
- **Generic recorder program** to record all or a subset of streams on the network
- **Generic stream viewer** programs (standalone and for MATLAB)
- **Open-source client programs** for a range of acquisition devices (EEG, MoCap, Sound, Video, Eyetracking, Human Interface Devices)
- Available separately: plugins for BCILAB, EEGLAB, MoBILAB

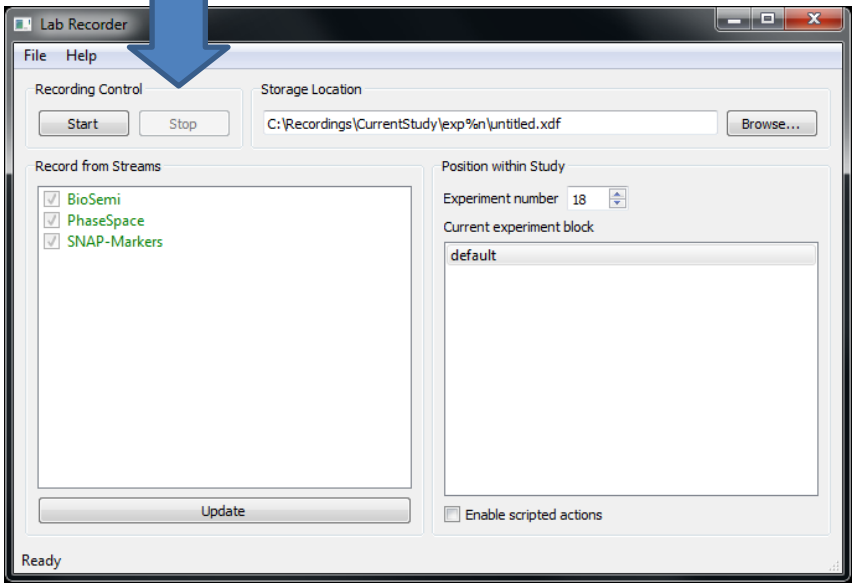
Currently Supported Hardware

- **EEG**: Biosemi, Cogionics, MINDO, BrainProducts, g.USBamp, Emotiv, Micromed, MindMedia, OpenEEG, TMSi, ANT Neuro ASALAB
- **Eye Tracking:** SR Research EyeLink, custom 2-camera setup
- **Motion Capture:** PhaseSpace, OptiTrack, Kinect, AMTI Force Plates
- **Human-Interface Devices:** Mice, Keyboards, Trackballs, Game Controllers, Wiimote and Expansions
- **Multimedia Devices:** PC-compatible sound cards, DirectShow-compatible video hardware
- **Untested:** ABM B-Alert, Enobio, Neuroscan Synamp, EGI AmpServer, Mitsar EEG, CTF/VSM, Tobii, SMI iViewX

Brief Usage Demo







XDF File Format

- Developed with Clemens Brunner (Graz Univ.)
- Independent of LSL, but supports full feature set (and comes with importers for MATLAB, EEGLAB, BCILAB, MoBILAB)
- Very simple (ca. 100 LoC parser) modern container file format supporting:
  - Any number of streams, time-synched
  - Extensible meta-data per stream with core subset specified online (code.google.com/p/xdf)

XDF Meta-Data Sample

- A portion of the Mocap meta-data specs:

```
<channels>            # specification of the channel layout
  <channel>           # information about a single channel (repeated for each)
    <label>           # label of the channel
    <marker>          # label of the marker that this channel refers to, if any
    <object>          # label of the object that this channel refers to, if any
    <type>            # type of data in this channel, can be an of the following values:
                      #  * PositionX, PositionY, PositionZ for euclidean position (strongly preferred unit: meters),
                      #  * OrientationA, OrientationB, OrientationC, OrientationD for quaternion-based orientations,
                      #  * Confidence for confidence information (preferred unit: normalized)
    <unit>            # measurement unit (e.g., meters)
  </channel>
</channels>


<acquisition>         # information about the acquisition system
  <manufacturer>      # manufacturer of the system
  <model>             # model name of the system
  <settings>          # settings of the acquisition system
  </settings>
  <compensated_lag>   # amount of hardware/system lag that has been implicitly
                      # compensated for in the stream's time stamps (in seconds)
</acquisition>


<setup>               # information about the physical setup (e.g. room layout)
  <name>              # name of the setup


 <bounds>             # bounding box of the space/room (in the same coordinate system as all others)
    <minimum>         # smallest possible position in the operating volume (for each axis)
      <X>
      <Y>
      <Z>
    </minimum>
    <maximum>         # largest possible position in the operating volume (for each axis)
      <X>
      <Y>
```

3 The Simulation and Neuroscience Application Platform (SNAP)

ftp://sccn.ucsd.edu/pub/SNAP

Purpose

- Allow to *rapidly prototype* game-like human-computer interactions with significant complexity
- *Generalize and advance* basic neuroscience experiments *towards practically relevant applications*
- Full source code, *no license restrictions on academic or commercial use and deployment*

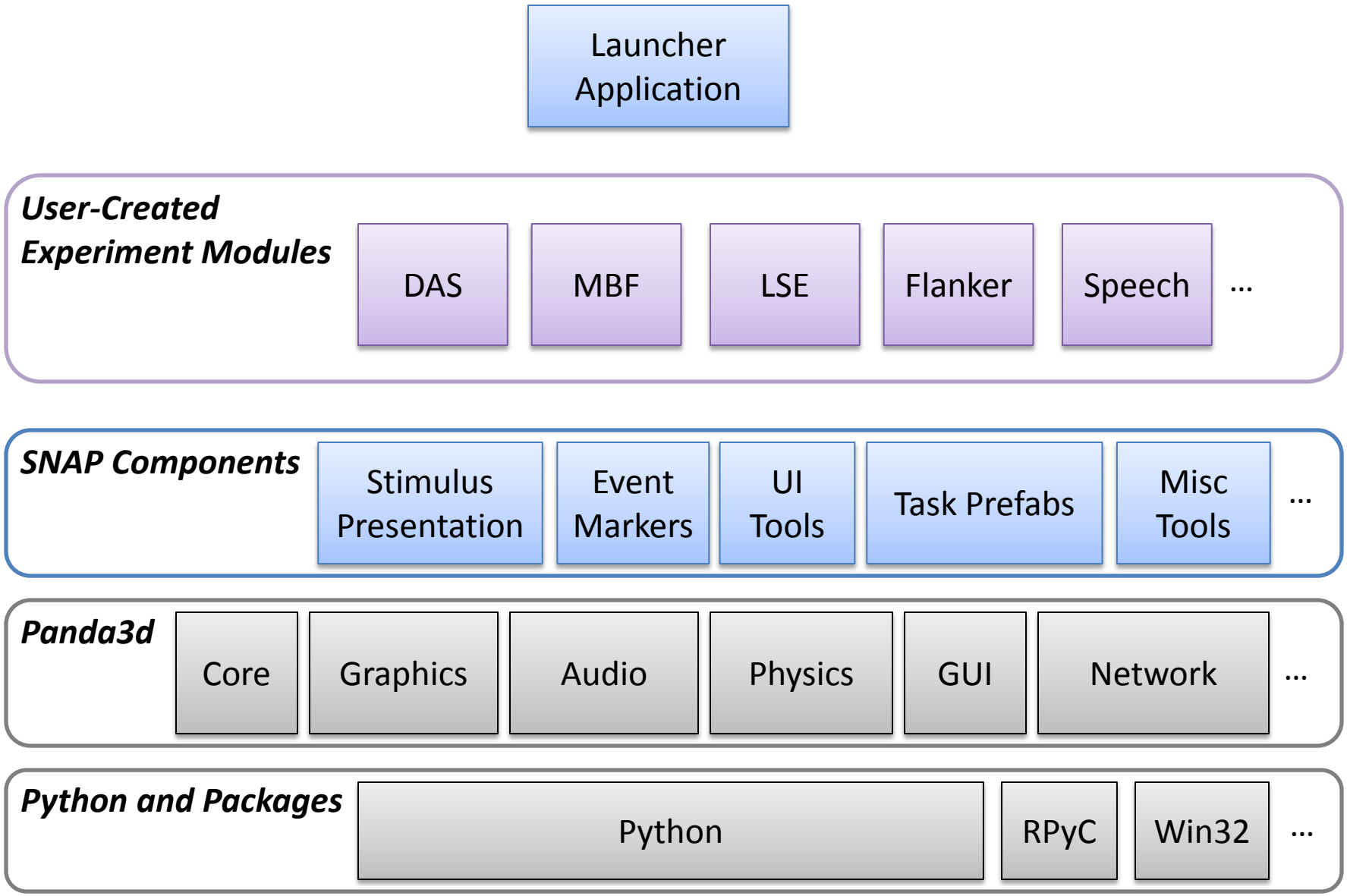
Approach

- Relies on **Python** as the scripting language and leverages its packages
- Uses the **Panda3d** game engine for graphics, audio, input, physics, GUI and low-level real-time subsystems
- Adds a thin **layer for experiment scripting**
- Adds some **extra low-level subsystems** (LSL, RPC, Pathfinding, …)

The Panda3d Engine

- **Con:**
  - Relatively clean but dated core (’97) but still actively developed, messy features at the outer fringes
  - Limited support for in-engine editing
  - no modern lighting/rendering model
- **Pro:**
  - Complete game engine, formerly commercial (Disney), now open source (MIT license) and maintained by CMU
  - Written in C++ (fast) and scriptable via Python (convenient)
  - Very comprehensive feature set for game/simulation purposes (750k LOC)
  - Remarkably good documentation (panda3d.org)

SNAP Architecture



Swartz Center for Computational Neuroscience

Basic Scripting

```
from framework.latentmodule import LatentModule
import random

class Main(LatentModule):
    def __init__(self):
        LatentModule.__init__(self)

        # set defaults for some configurable parameters:
        self.num_trials = 50        # number of trials in first part
        self.text_probability = 0.5 # probability that a text is displayed instead of a picture

    def run(self):
        self.marker(10)  # emit an event marker to indicate the beginning of the experiment
        self.write('This is a sample experiment.\nYou will be lead through a few trials in the f
        self.write('Press the space bar when you are ready.','space')

        for k in range(self.num_trials):
            # show a 3-second cross-hair
            self.crosshair(3)
            # display either a text or a picture
            if random.random() < self.text_probability:
                self.marker(1)
                self.write('A text.',scale=0.5)
            else:
                self.marker(2)
                self.picture('monkey.jpg',2,scale=0.3)
            # wait for 2 seconds
            self.sleep(2)

        self.sound('nice_bell.wav',volume=0.5)
        self.write('You successfully completed the experiment.')
```

Complex Scripting

- Example: earlier experiment prototype (MBF)

Relationship to BCILAB and LSL

- Natively sends event marker streams to LSL
- Can be remotely controlled by the LSL LabRecorder experiment control features (e.g., load/config/start/stop)
- BCILAB can remotely control the value of any set of module parameters online (for example, the “task load level”)
- Can also read/write any LSL stream manually through the Python API

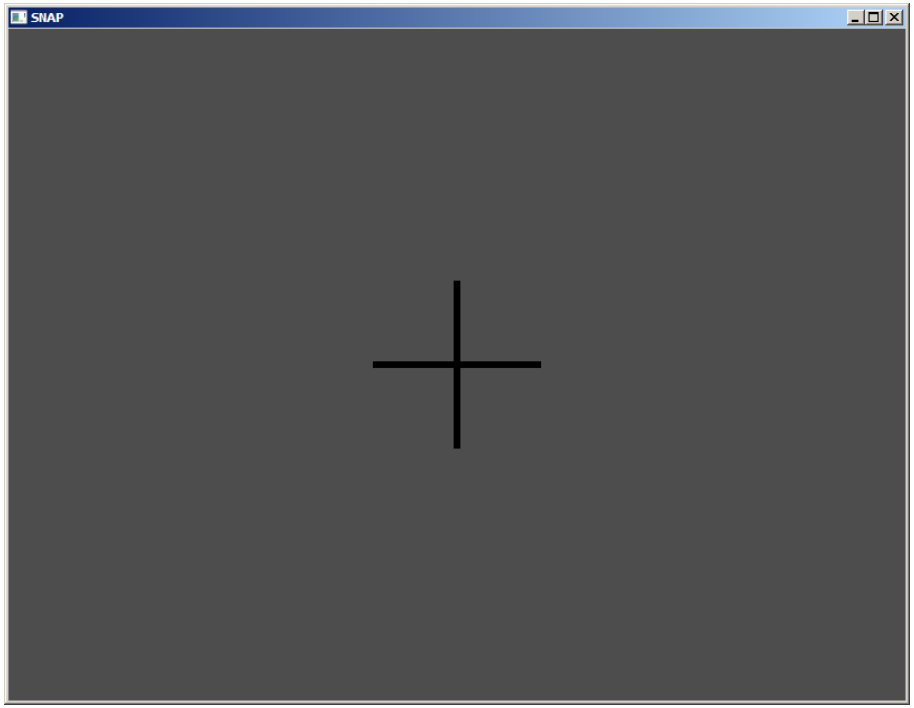
Benefits Summary

- Basic scripting is *as simple as it can get*
- Scales gracefully from basic *to far more complex* experiments, both in terms of features and performance
- Integrates painlessly with LSL and BCILAB

Caveats

- Stimulus presentation not necessarily with same hard timing guarantees as traditional neuroscience applications
- Lacking rich authoring tools (e.g., dataflow graphs) of some commercial software, instead relies fully on scripting and external authoring tools (3ds max, Eclipse)
- Live coding/debugging not yet as effortless as it could be (requires a very disciplined workflow), also fairly long loading/iteration times for complex experiments (>1 minute)

Brief Demo



4 The BCILAB Toolbox

http://sccn.ucsd.edu/**wiki**/**BCILAB**
ftp://sccn.ucsd.edu/pub/bcilab

Software Environment For:

- **Brain-Computer Interface Design** (Cognitive Monitoring)
- **Methods Research**:
  - Design & rapid prototyping of new methods & methods from literature
  - Offline testing, performance evaluation & batch comparison, visualizations
  - Simulated online testing
- **Rapid Prototyping**:
  - Real-time use and testing of BCIs
  - Prototype deployment

Facts & Figures

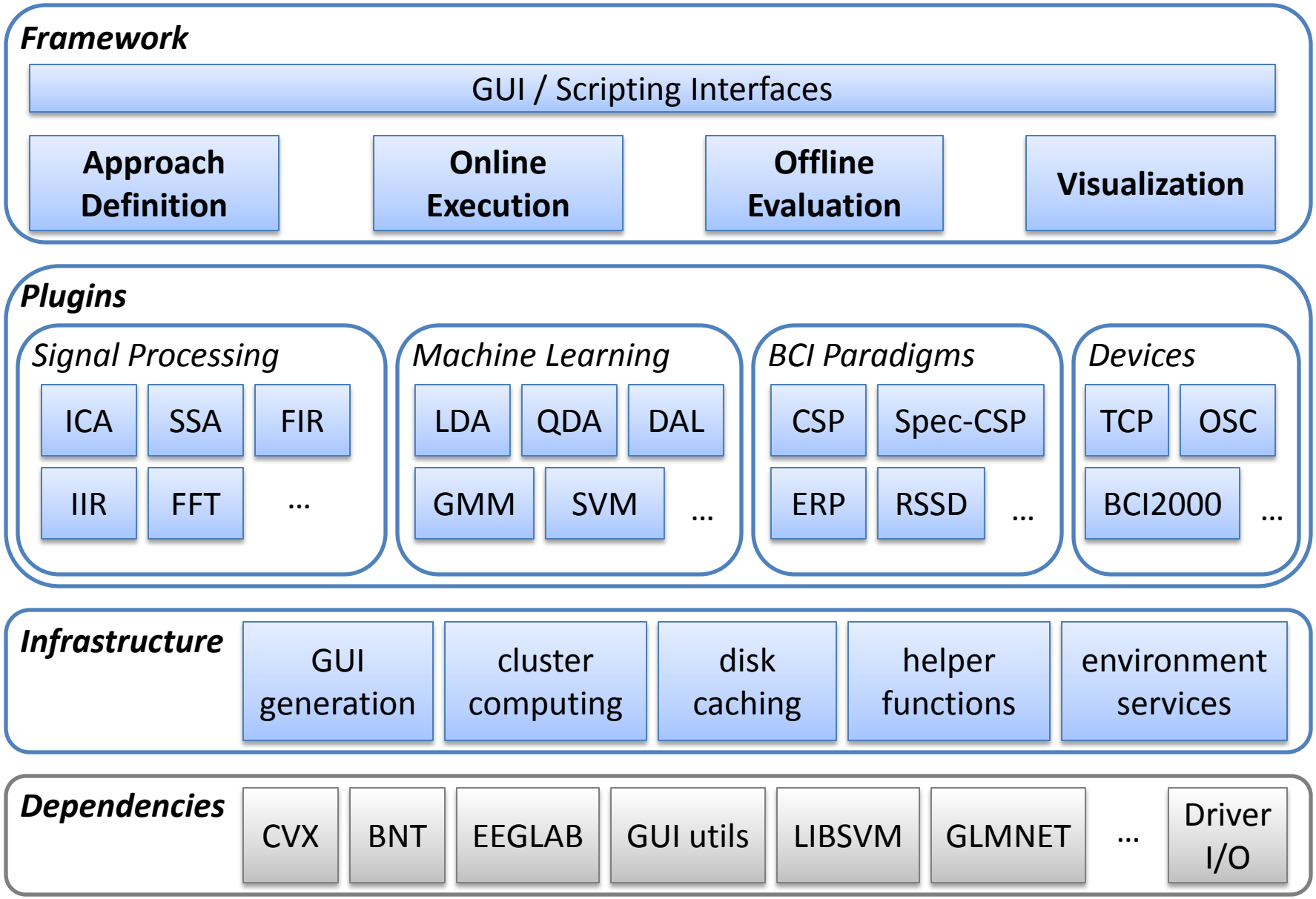
- Developed since 2010 at SCCN, UCSD (primarily by me)
- Precursor was the PhyPA toolbox (Kothe & Zander, 2006-'09)
- Built on top of EEGLAB (Delorme & Makeig, 2004)
- The largest open-source BCI toolbox by methods and algorithms (100+) as of 2011
- Offline and online processing both in MATLAB, same code base, Win/Linux/MacOS, 32/64bit
- Extensive documentation (hundreds of pages of help text, manual, wiki, 400+ lecture slides online)

Office of Naval Research

Swartz Center for Computational Neuroscience

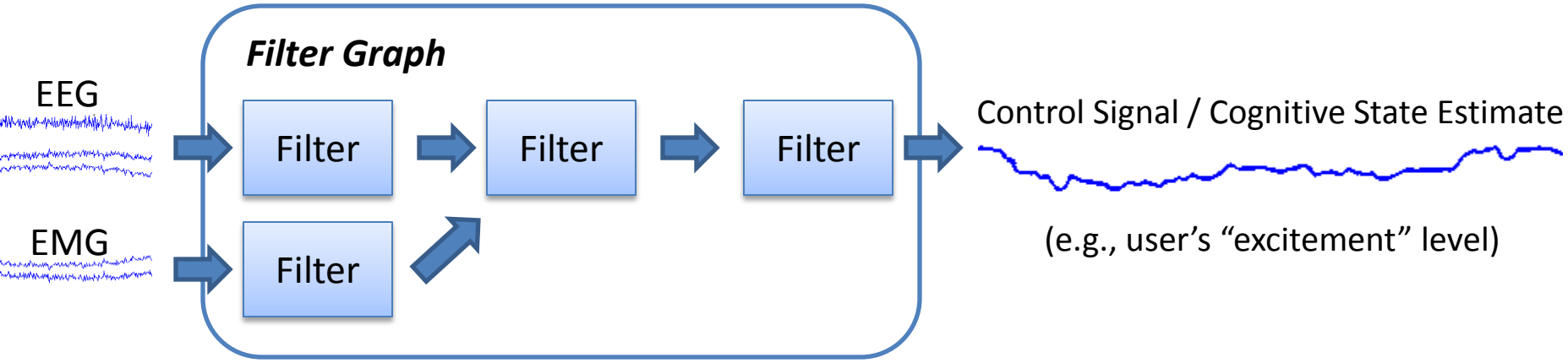
4.1 Toolbox Overview

Architectural Overview



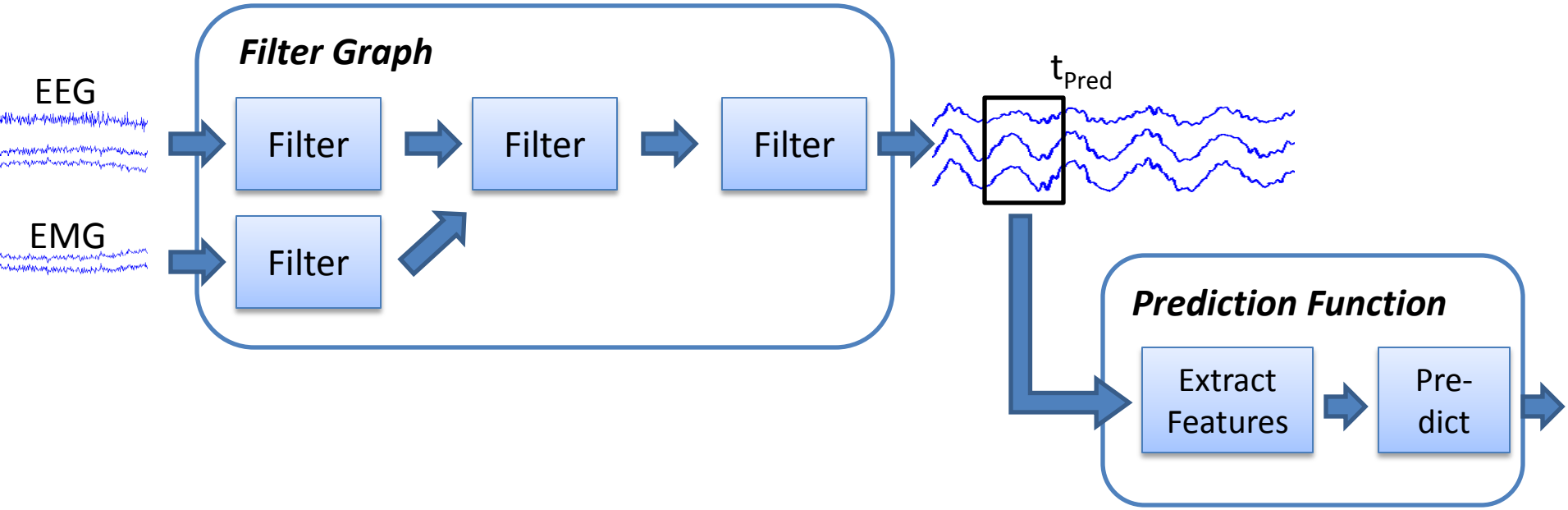
Functional Overview: Online Processing

- Supports BCI designs framed as a series of *Signal Processing Blocks* or as a *Prediction Function*, or a *combination of both*



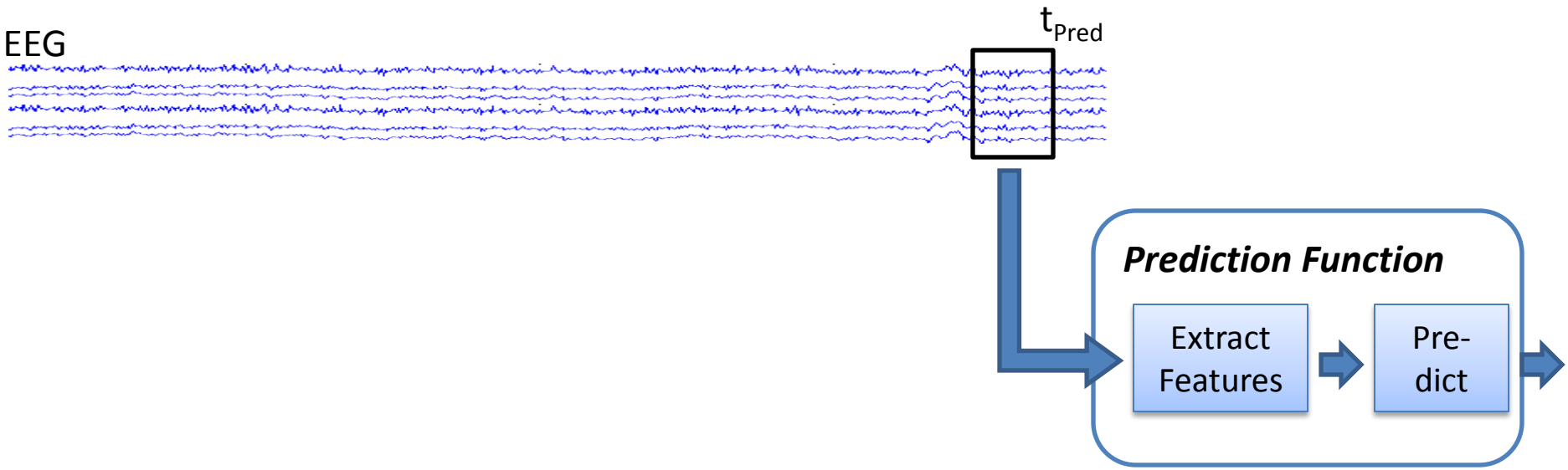
Functional Overview: Online Processing

- Supports BCI designs framed as a series of *Signal Processing Blocks* or as a *Prediction Function*, or a *combination of both*

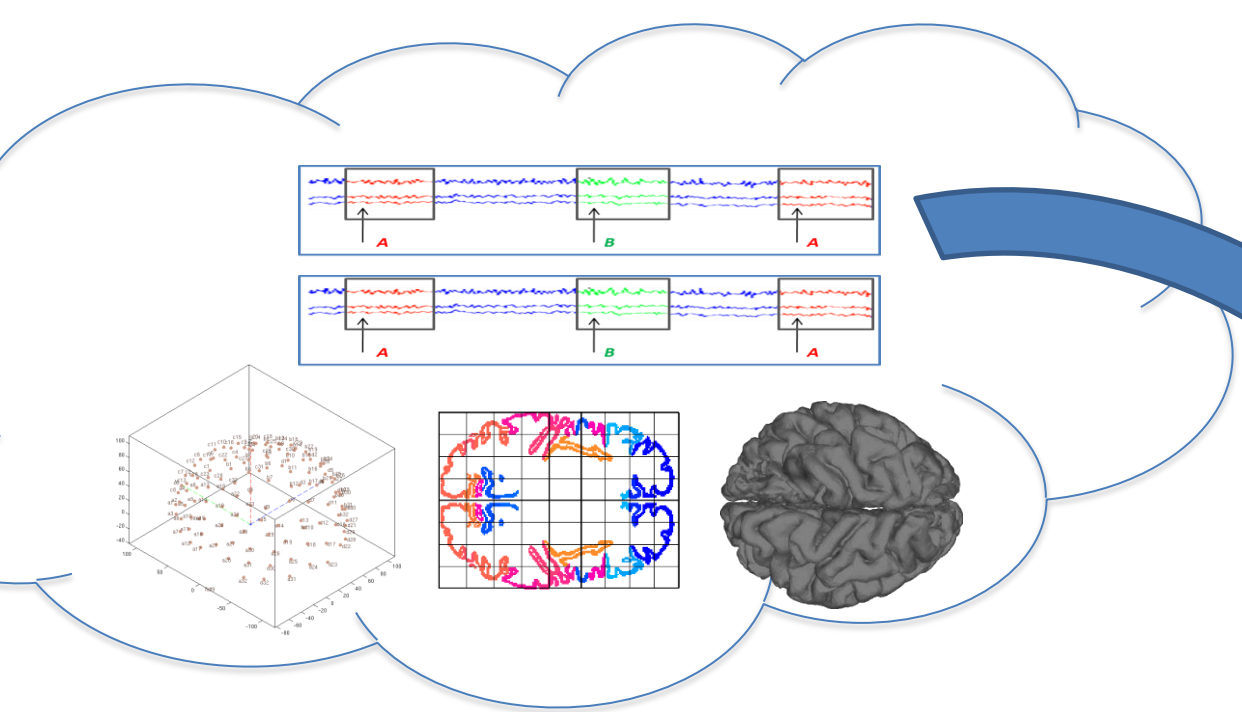


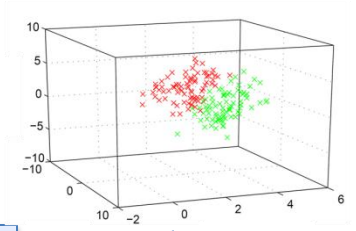
Functional Overview: Online Processing

- Supports BCI designs framed as a series of *Signal Processing Blocks* or as a *Prediction Function*, or a *combination of both*



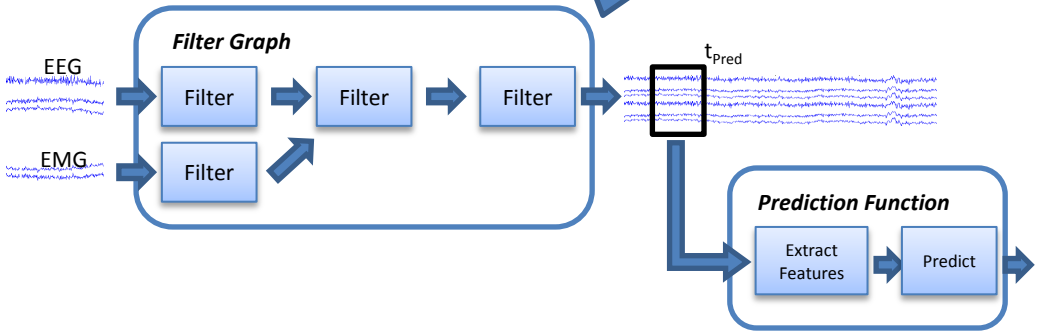
Functional Overview: Calibration





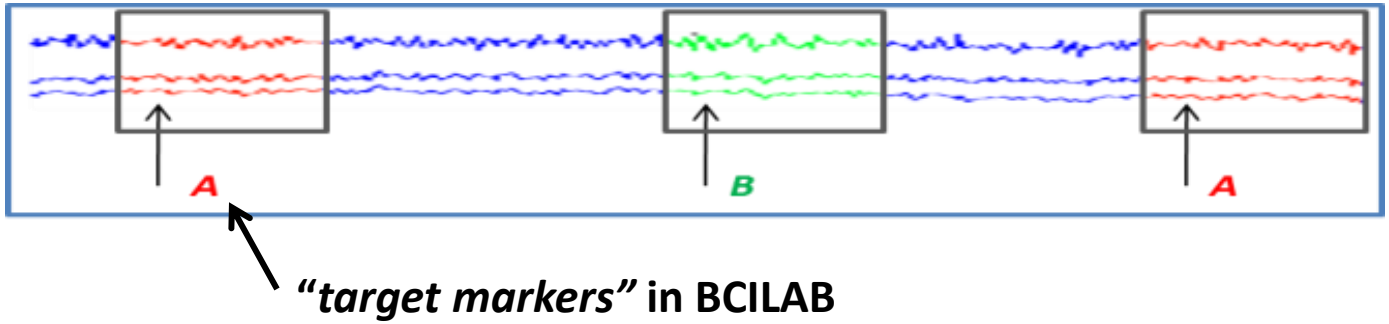
Learning Function

BCI model



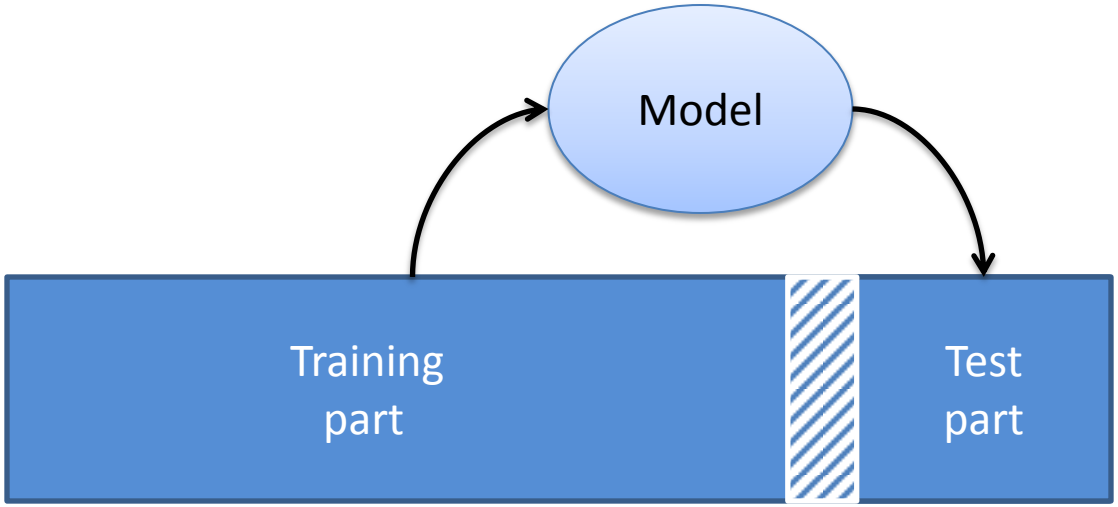
Calibration Data Reminder

- Typical features of a calibration recording:
  - continuous EEG (or other)
  - multiple trials/blocks (capturing variation)
  - randomized (eliminating confounds)
  - event markers to encode cognitive state conditions of interest, e.g., stimuli/responses (called "*target markers*" in BCILAB)



Functional Overview: Evaluation

- **Covers:**
  - Simulated online processing
  - Cross-validation (shown below)
  - Grid Search
  - Nested Cross-Validation



Swartz Center for Computational Neuroscience

4.2 Workflows and Concepts

Pipeline Notion

- BCILAB is a framework that resembles a processing pipeline: first configure everything, then apply it to one or more data sets
- **Configuration Inputs:**
  - Base BCI Paradigm to execute – "what to run?"
  - Custom parameters for the paradigm
  - Evaluation Scheme – "how to run it?" (e.g., what type of cross-validation)
  - Definition of the dataset annotations (mapping between event marker strings and class labels)

Pipeline Processes

- **Curate:** bring the input data into standard form
- **Design:** define the computational approach
- **Train:** invoke all steps necessary for training (calibrating) a BCI and estimates performance
- **Predict/Evaluate:** apply a BCI to some data offline
- **Visualize:** visualize BCI model internals
- **Run Online:** apply a BCI online / incrementally
- **Batch Analysis:** perform a series of processing steps, optionally in parallel

A Note on Data Curation

- Up-front conversion of data set and file format into *uniform representation*:
  - Continuous data – unfiltered
  - Correct channel labels/locations
  - Correct event types, latencies, etc
  - Other common meta-data about raw recordings
- Usually done in a first pass before any BCILAB function is touched

Plugin Concepts: Filters

- Filters can operate on continuous signals…

- … or on segmented (“epoched”) signals:

Plugin Concepts: Filters

- *Static ("stateless") filters*:
  ```
  EEG = flt_selchans(EEG,{'C3','C4','Cz'})
  ```

- *Dynamic ("stateful") filters*:
  ```
  [EEG,State] = flt_resample(EEG,200,State)
  ```

- *Epoched filters*:
  ```
  EEG = flt_fourier(EEG)
  ```

Plugin Concepts: Filters

- **Caveat:** filters have *lazy evaluation behavior*, i.e. they do not evaluate unless forced:

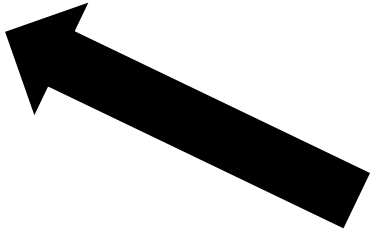
```
EEG = flt_fourier(EEG)
>> EEG =
      head: @flt_fourier
     parts: {[1x1 struct]}
  codehash: '356d73563c38107c63a33762cc7789ba'
```

Not what you wanted!

Plugin Concepts: Filters

- **Caveat:** filters have lazy evaluation behavior, i.e. they do not evaluate unless forced:

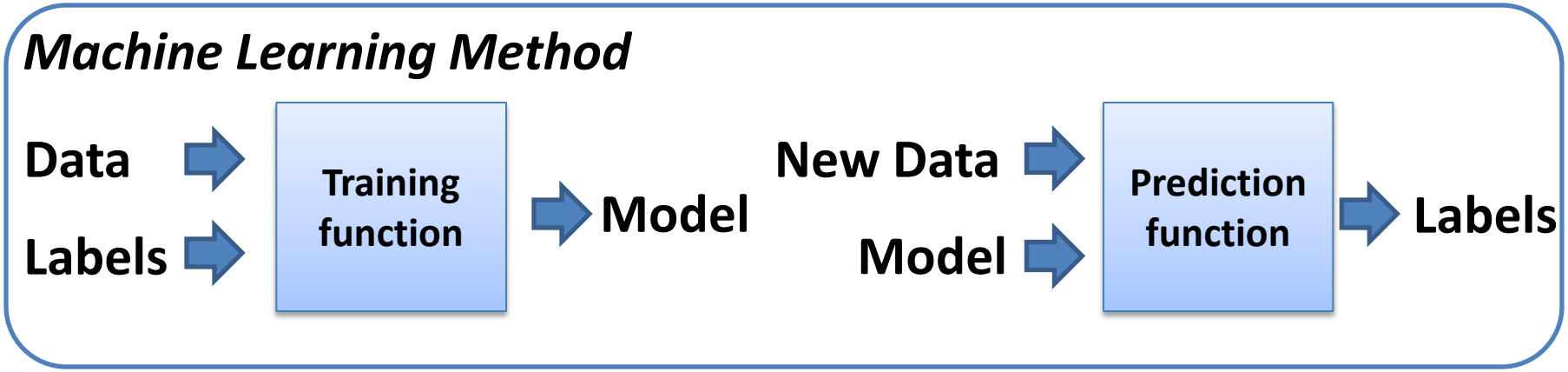
```
EEG = exp_eval(flt_fourier(EEG))
```



The right way

Plugin Concept: Machine Learning

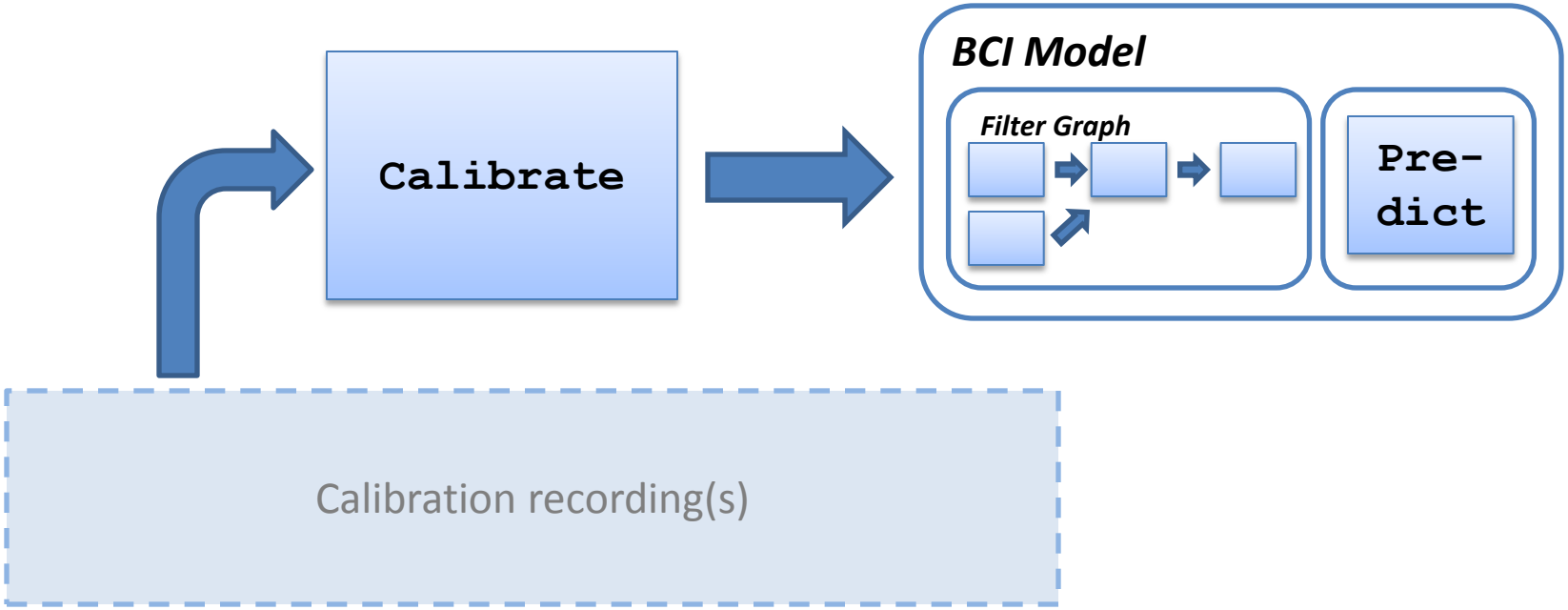
- Machine learning functions come in pairs:



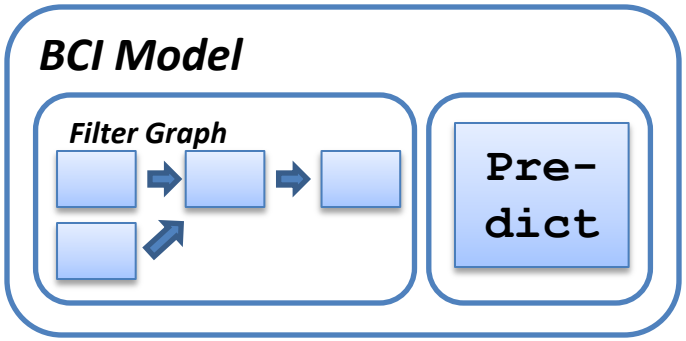

```
M = ml_trainlda(X,y)
p = ml_predictlda(Xnew,M)
```

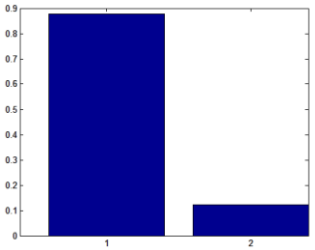
Plugin Concepts: Paradigms

- **BCI paradigms** are the coarsest plugin type in BCILAB and *tie all parts of a BCI approach together* (signal processing, feature extraction, machine learning, …)
- They are invoked by the offline/online framework



Data Representations

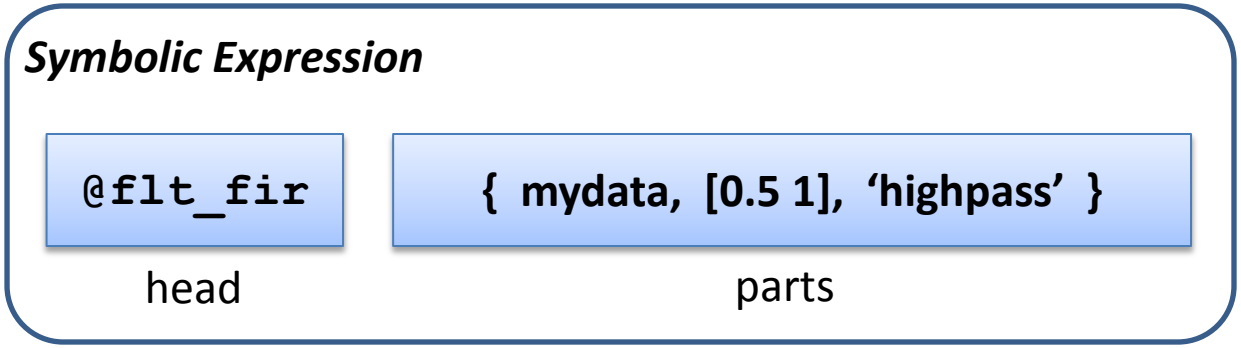




Probability Distributions

$$\begin{bmatrix} f_1 \\ f_2 \\ \vdots \end{bmatrix}$$

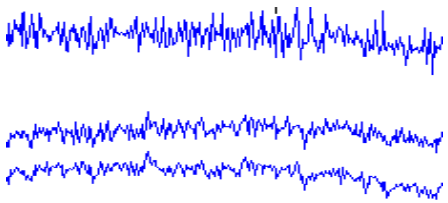
Feature Vectors



Data Representations

Signal

.data

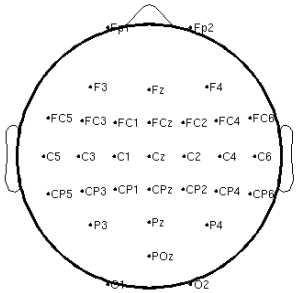


.event

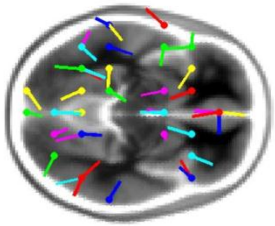
.srate *200Hz*

.xmin *0.0s*

.chanlocs



.dipfit



... (meta-data)

Signal Bundle

.streams

- Signal 1
- Signal 2
- ⋮
- Signal n

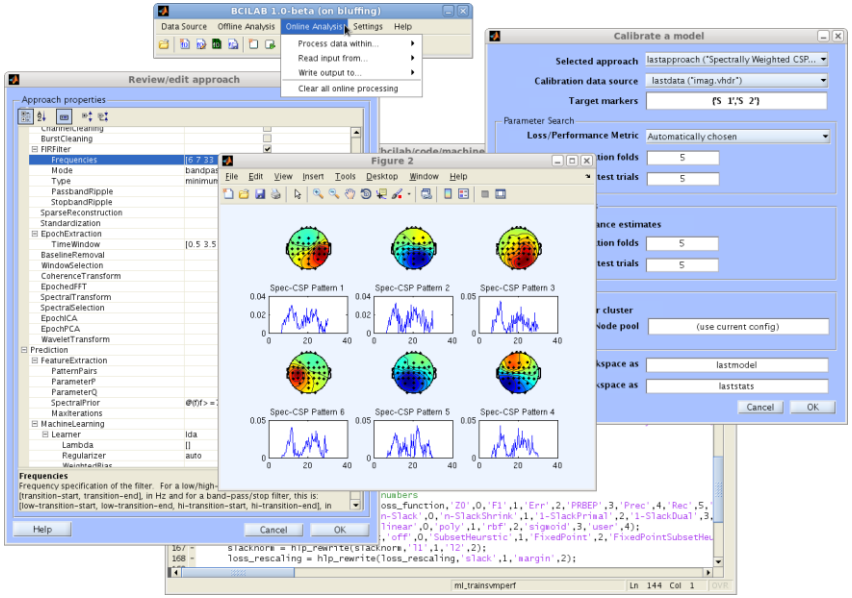
... (meta-data)

Dataset Collection

- Bundle 1
- Bundle 2
- ⋮
- Bundle n

4.3 In-Depth Walkthrough

GUI/Script Walkthrough




```
epoch = [-0.2 0.8];
wnds = [0.25 0.3;0.3 0.35;0.35 0.4; 0.4 0.45;0.45 0.5;0.5 0.55;0.55 0.6];

apps.wmeans_lda = {'Windowmeans' 'SignalProcessing',{'IIRFilter',{[0.1 0.5],'highpass'}, ...
    'EpochExtraction',epoch,'SpectralSelection',[0.1 15]},'Prediction',{'FeatureExtraction',{'wnds',wnds}}};
apps.wmeans_vblogreg = {'Windowmeans' 'SignalProcessing',{'IIRFilter',{[0.1 0.5],'highpass'}, ...
    'EpochExtraction',epoch,'SpectralSelection',[0.1 15]},'Prediction',{'FeatureExtraction',{'wnds',wnds}, ...
    'MachineLearning',{'Learner',{'logreg',[],'variant','vb-iter'}}}};
apps.dalfine = {'DALERP','SignalProcessing',{'EpochExtraction',epoch}, ...
    'Prediction',{'MachineLearning',{'Learner',{'dal','lambdas',2.^(10:-0.125:1),'solver','cg'}}}};
apps.raw_glc = {'DataflowSimplified' 'SignalProcessing',{'IIRFilter',{[0.1 0.5],'highpass'}, ...
    'EpochExtraction',epoch,'SpectralSelection',[0.1 15]}, ...
    'Prediction',{'MachineLearning',{'learner',{'dal',2.^(12:-0.125:1),'regularizer','glc', 'shape',[256 NaN]}}}};
apps.wavelet_glc = {'DataflowSimplified' 'SignalProcessing',{'IIRFilter',{[0.1 0.5],'highpass'}, ...
    'EpochExtraction',epoch,'SpectralSelection',[0.1 15],'wavelet','on'}, ...
    'Prediction',{'MachineLearning',{'learner',{'dal',2.^(12:-0.125:1),'regularizer','glc', 'shape',[256 NaN]}}}};

results = bci_batchtrain('Data','/data:/grainne/ERN/*.vhdr','Approaches',apps, ...
    'TargetMarkers',{{'S101','102'},{'S201','202'}});
```

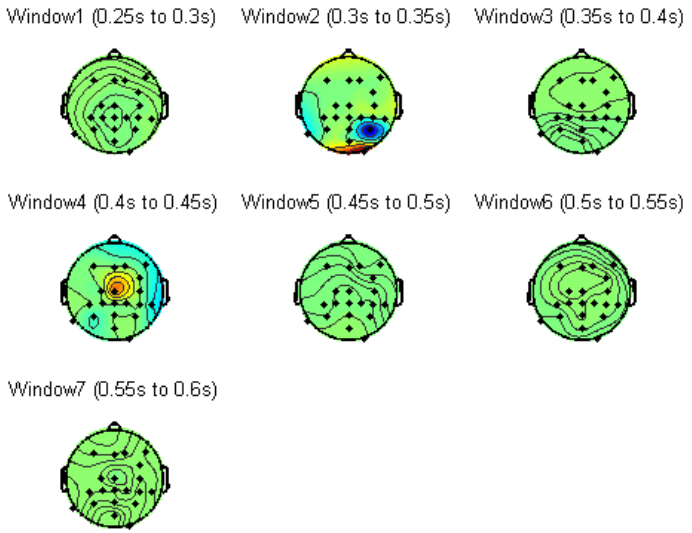
ERP Sample Task

- **Flanker Task:** The experiment consists of a sequence of ca. 330 trials with inter-trial interval of 2s +/- 1.5s
- In each trial, an arrow is presented centrally (pointing either left or right)
- The arrow is flanked by congruent or incongruent "flanker" arrows (preceding the center by a few ms):

- The **subject is asked to press the left or right button, according to the central arrow direction, and makes frequent errors (ca. 25%)**

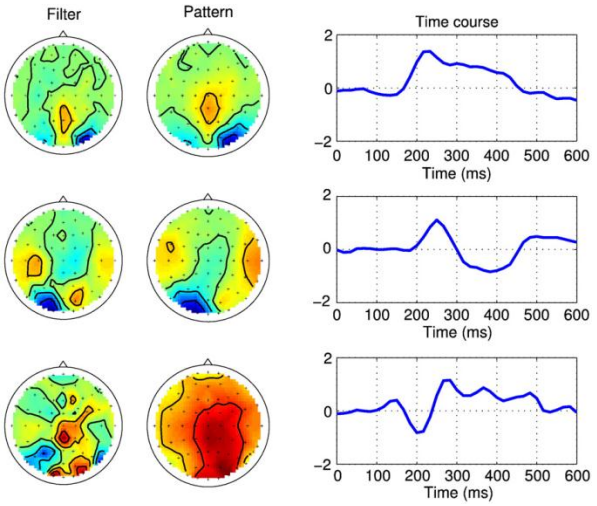
Time-Domain / ERP Baseline

Windowed Means



- Traditional linear classifier for event-locked brain responses, usually using LDA
- Time windows manually assigned
- Examples: error recognition, surprise

DAL-ERP



(*)

- State-of-the-art approach, no hand-tuned parameters
- Uses rank-regularized logistic or linear regression

Note: some theory-focused slides available at

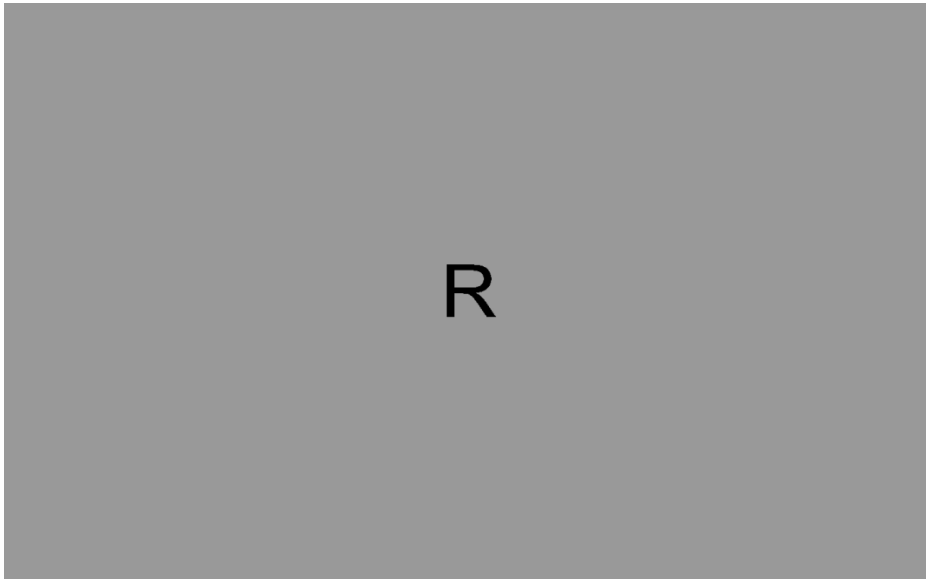
ftp://sccn.ucsd.edu/pub/bcilab/lectures/05%20ERP%20Processing.pdf

ftp://sccn.ucsd.edu/pub/bcilab/lectures/08%20Optimization-based%20Approaches.pdf

(*image: Tomioka et al., 2010)

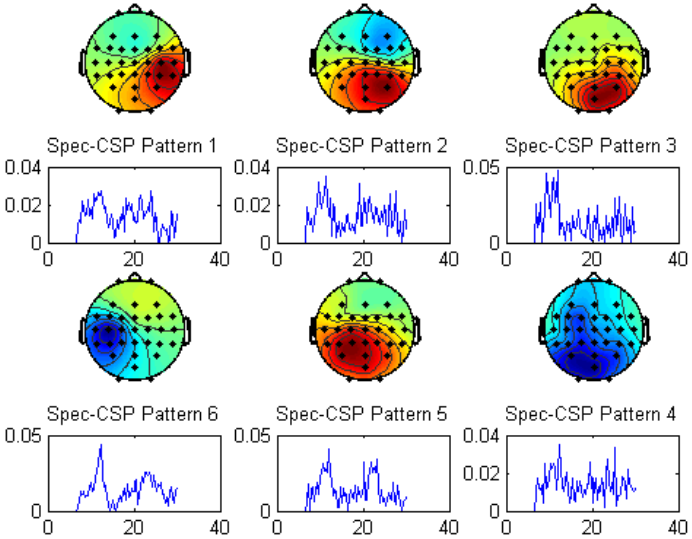
Oscillatory-Process Sample Task

- The experiment consists of 160 trials (pause at ½ the experiment) . Each trial begins with a letter (either L or R) displayed for 3s. The subject is instructed to subsequently imagine either a left-hand or a right-hand movement. Each trial ends with a blank screen displayed for 3.5s.



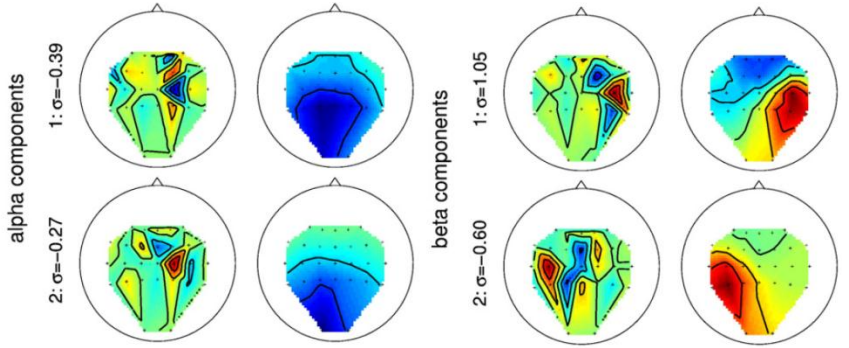
Oscillatory Processes Baseline

Common Spatial Patterns Family



- Filter-Bank CSP (FBCSP): multiple bands
- Diagonal Loading CSP (DLCSP): cov. shrinkage
- Composite CSP (CCSP): covariance prior
- Tikhonov-regularized CSP (TRCSP): filter shrinkage
- Spectrally weighted CSP (Spec-CSP): learning spectral filters from the data

DAL-OSC



(*)

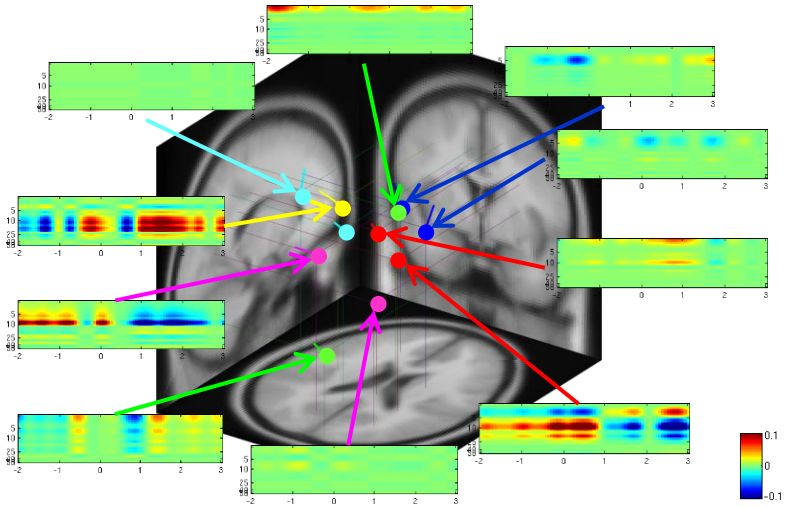
- State-of-the-art approach, no hand-tuned parameters
- Also uses rank-regularized logistic or linear regression
- Single-step approach, jointly optimal

ftp://sccn.ucsd.edu/pub/bcilab/lectures/07%20Oscillatory%20Processes.pdf

(*image: Tomioka et al., 2010)

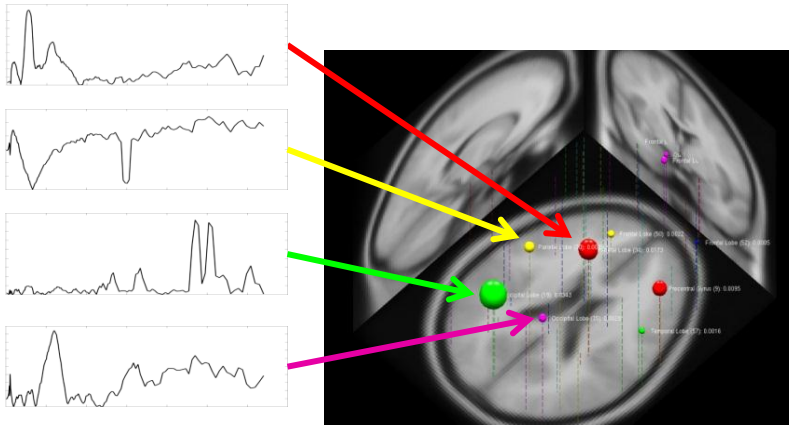
New Methods

Regularized Spatio-Spectral Dynamics



- Applicable to slowly-changing operator state and background activity as well as event-related transients
- RSSD is a pioneering method for learning full source-level time/frequency structure
- Examples: cognitive load, attention shifts
- Presented at ICON'11; methods and data papers in preparation

Multi-subject Overcomplete Spectral Regression



- Long-term stationary oscillations
- Can integrate information from a corpus of data (across persons)
- Examples: fatigue, alertness, sleep stages
- Presented at EMBC'11
- Related method presented at ABCI'11

Swartz Center for Computational Neuroscience

4.4 Adding New Methods

Swartz Center for Computational Neuroscience

Adding New Methods

```
arg_define([0 3],varargin, ...
    arg_norep('trials'), ...
    arg_norep('targets'), ...
    arg({'cost','Cost'}, search(2.^(-5:2:15)), [], 'Regularization parameter. Reasonable range: 2.^(-5:2:15), greater is stronger. By default, it is average
    arg({'ptype','Type'}, 'classification', {'classification','regression','ranking'}, 'Type of problem to solve.','cat','Core Parameters'), ...
    arg({'kernel','Kernel'}, 'rbf', {'linear','rbf','poly','sigmoid','user'}, 'Kernel type. Linear, or Non-linear kernel types: Radial Basis Functions (gene
    arg({'g','RBFScale','gamma'}, search(2.^(-16:2:4)), [], 'Scaling parameter of the RBF kernel. Should match the size of structures in the data; A reasonal
    arg({'d','PolyDegree'}, uint32(3), [], 'Degree for the polynomial kernel.','cat','Core Parameters'), ...
    arg({'etube','EpsilonTube','tube'}, 0.1, [], 'Epsilon tube width for regression.','cat','Core Parameters'), ...
    arg({'rbalance','CostBalance','balance'}, 1, [], 'Relative cost of per-class errors. The factor by which training errors on positive examples outweight
    ...
    arg({'s','SigmoidPolyScale'}, 1, [], 'Scale of sigmoid/polynomial kernel.','cat','Miscellaneous'), ...
    arg({'r','SigmoidPolyBias'}, 1, [], 'Bias of sigmoid/polynomial kernel.','cat','Miscellaneous'), ...
    arg({'u','UserParameter'}, '1', [], 'User-defined kernel parameter.','cat','Miscellaneous','type','char','shape','row'), ...
    arg({'bias','Bias'}, false, [], 'Include a bias term. Only implemented for linear kernel.','cat','Miscellaneous'), ...
    arg({'scaling','Scaling'}, 'std', {'none','center','std','minmax','whiten'}, 'Pre-scaling of the data. For the regulariation to work best, the features
    arg({'clean','CleanUp'}, false, [], 'Remove inconsistent training examples.','cat','Miscellaneous'), ...
    arg({'epsi','Epsilon','eps'}, 0.1, [], 'Tolerated solution accuracy.','cat','Miscellaneous'), ...
    arg({'verbose','Verbose'}, false, [], 'Show diagnostic output.','cat','Miscellaneous'));

if is_search(cost)
    cost = 1; end
if is_search(g)
    g = 0.3; end

% find the class labels
classes = unique(targets);
if length(classes) > 2
    % in this case we use the voter
    model = ml_trainvote(trials,targets,'1v1',@ml_trainsvmlight,@ml_predictsvmlight,varargin{:});
else
    % scale the data
    sc_info = hlp_findscaling(trials,scaling);
    trials = hlp_applyscaling(trials,sc_info);

    % remap target labels to -1,+1
    targets(targets==classes(1)) = -1;
    targets(targets==classes(2)) = +1;

    % rewrite sme string args to numbers
    ptype = hlp_rewrite(ptype,'classification','c','regression','r','ranking','p'); %#ok<*NODEF>
    kernel = hlp_rewrite(kernel,'linear',0,'poly',1,'rbf',2,'sigmoid',3,'user',4);

    % build the arguments
    args = sprintf('-z %s -c %f -v %d -w %f -j %f, -b %d -i %d -e %f -t %d -d %d -g %f -s %f -r %f -u %s', ...
        ptype,cost,verbose,etube,rbalance,bias,clean,epsi,kernel,d,g,s,r,u);

    % run the command
    model = svmlearn(trials,targets,args);
    model.sc_info = sc_info;
    model.classes = classes;
end
```

5 Further Reading

Documentation Resources

- **Presentations:** ftp://sccn.ucsd.edu/pub/bcilab/presentations/
- **Lecture:** ftp://sccn.ucsd.edu/pub/bcilab/lectures/
- **Manuals:** ftp://sccn.ucsd.edu/pub/bcilab/manuals/
- **Wiki:** http://sccn.ucsd.edu/wiki/BCILAB
- **Function References:** bcilab-1.xx/build/docs/index.html
- **Release Notes:** bcilab-1.xx/RELEASE NOTES.TXT
- **Talk Videos:**
  https://www.youtube.com/watch?v=w8Z3b_aftco (part 1)
  https://www.youtube.com/watch?v=YUB0vxNmm2w (part 2)

Thanks!

Questions?