Colorado State University
Brain-Computer Interfaces Lab.

CEBL$_3$: A Modular Platform for EEG Signal Analysis and Real-Time Brain-Computer Interfaces

Elliott Forney and Charles Anderson

Grant Number 1065513.

Non-Invasive Brain-Computer Interfaces

Brain-Computer Interfaces (BCI) are systems for establishing a direct channel of communication between the human brain and a computerized device.

BCI utilize a communication protocol along with signal processing and machine learning algorithms to convey the user's intent to a computer system.

In our lab, we use Electroencephalography (EEG) to monitor brain activity because it is non-invasive, portable and realtively affordable.

An important application for BCI is in the development of assistive technologies for people with severe motor impairments.

For those who find it difficult to communicate or perform everyday tasks, even a somewhat slow BCI may prove to be an invaluable tool.

A New Software Platform for BCI

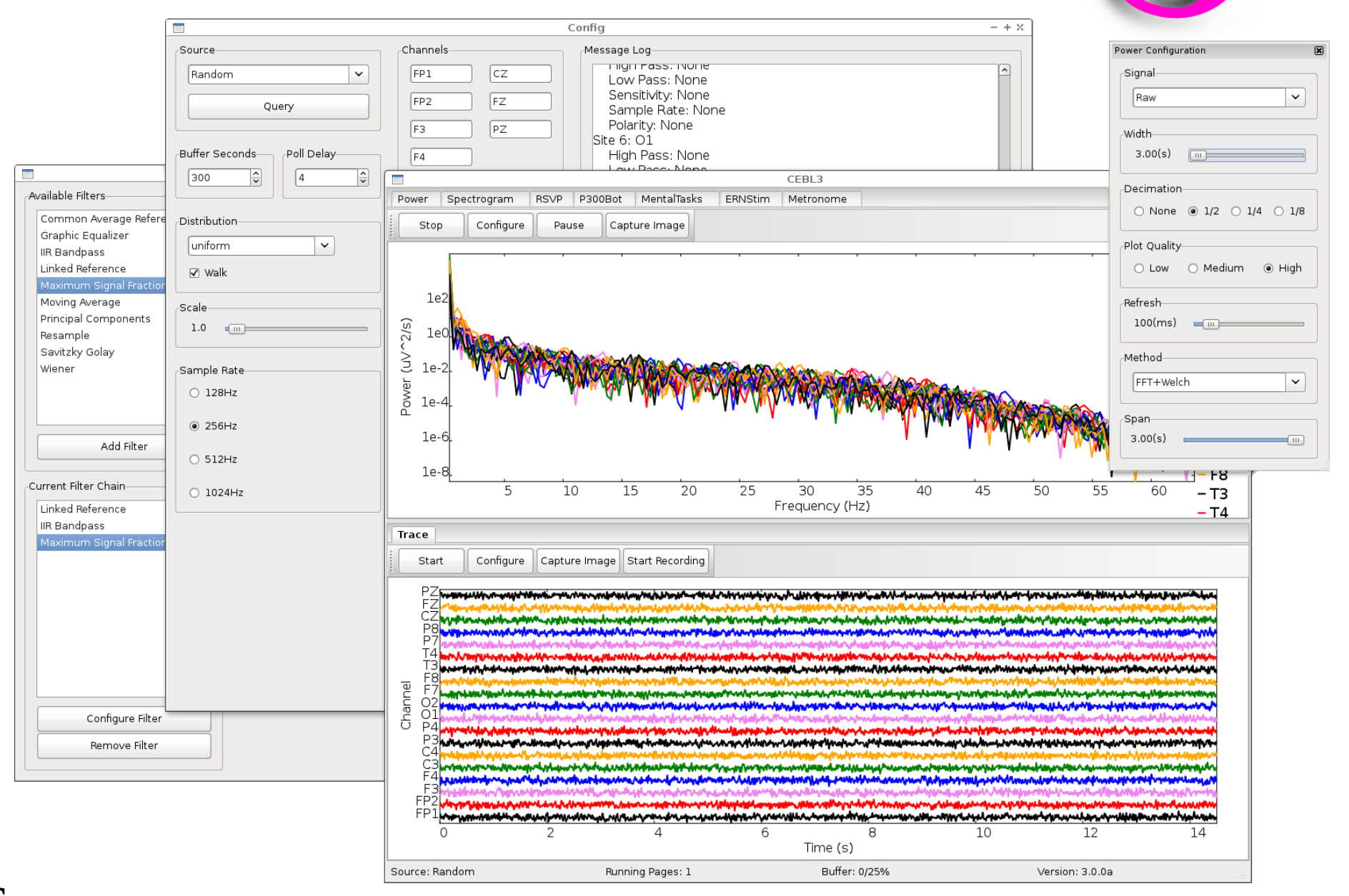
CEBL$_3$

CEBL$_3$ is a complete re-write of our BCI software platform.

Currently available BCI software systems fall short in one of two areas. Either:

-they are not well-suited for real-time experiments

-or they are not easily extended to use new paradigms and methods.



CEBL$_3$ addresses these issues through the use of high-level programming languages and a modular design approach that promotes code reuse throughout the entire process of developing BCI prototypes.

Additionally, CEBL$_3$ includes a number of state-of-the-art signal processing and machine learning algorithms as well as various modules for graphics, visualization and user interface development.

Python, et al.

CEBL$_3$ is written almost entirely in Python.

Python is a concise, dynamically-typed and interpreted programming language.

This allows for the rapid-development of clearly-written modules and API's.

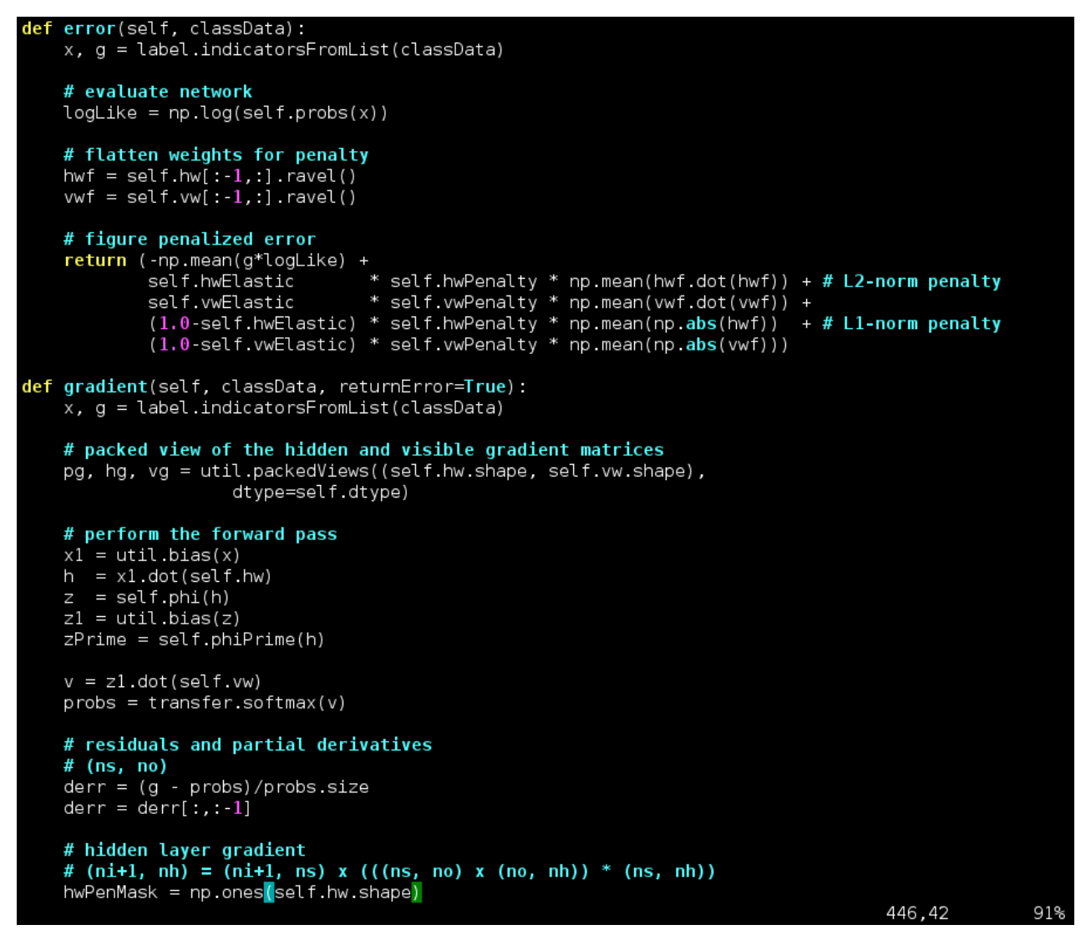
Python also has many available libraries and packages that allow developers to leverage existing functionality.

CEBL$_3$ relies heavily on the NumPy and SciPy libraries which utilize highly optimized C and Fortran libraries for numerical computations.

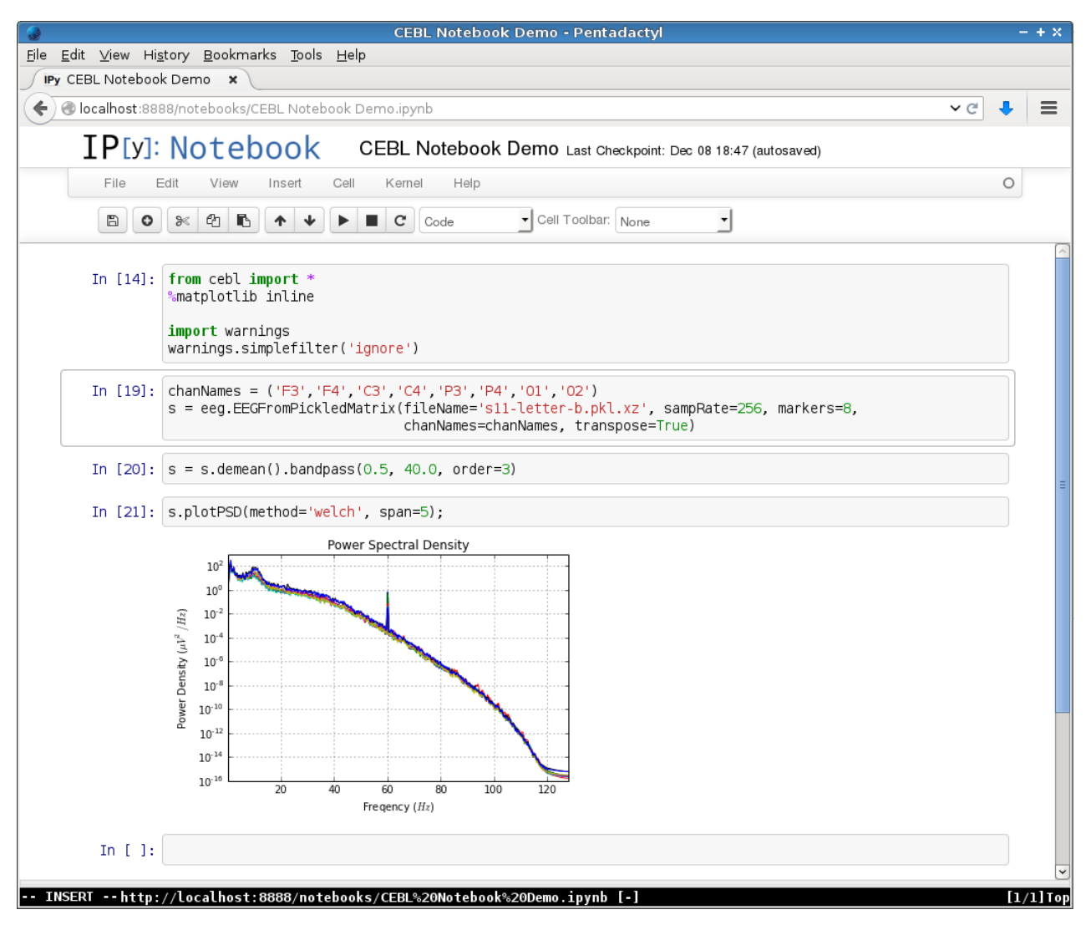
Matplotlib is also used to generate both static and real-time visualizations.

The wxPython library is used for graphics and user interfaces.

Custom modules or routines can also be written in C in order to maintain the levels of performance required for real-time processing.



```
def error(self, classData):
    x, g = label.indicatorsFromList(classData)

    # evaluate network
    logLike = np.log(self.probs(x))

    # flatten weights for penalty
    hwf = self.hw[:-1,:].ravel()
    vwf = self.vw[:-1,:].ravel()

    # figure penalized error
    return (-np.mean(g*logLike) +
            self.hwElastic       * self.hwPenalty * np.mean(hwf.dot(hwf)) + # L2-norm penalty
            self.vwElastic       * self.vwPenalty * np.mean(vwf.dot(vwf)) +
            (1.0-self.hwElastic) * self.hwPenalty * np.mean(np.abs(hwf))  + # L1-norm penalty
            (1.0-self.vwElastic) * self.vwPenalty * np.mean(np.abs(vwf)))

def gradient(self, classData, returnError=True):
    x, g = label.indicatorsFromList(classData)

    # packed view of the hidden and visible gradient matrices
    pg, hg, vg = util.packedViews((self.hw.shape, self.vw.shape),
                    dtype=self.dtype)

    # perform the forward pass
    x1 = util.bias(x)
    h  = x1.dot(self.hw)
    z  = self.phi(h)
    z1 = util.bias(z)
    zPrime = self.phiPrime(h)

    v = z1.dot(self.vw)
    probs = transfer.softmax(v)

    # residuals and partial derivatives
    # (ns, no)
    derr = (g - probs)/probs.size
    derr = derr[:,:-1]

    # hidden layer gradient
    # (ni+1, nh) = (ni+1, ns) x (((ns, no) x (no, nh)) * (ns, nh))
    hwPenMask = np.ones(self.hw.shape)
                                                                  446,42         91%
```



CEBL Notebook Demo - Pentadactyl

localhost:8888/notebooks/CEBL Notebook Demo.ipynb

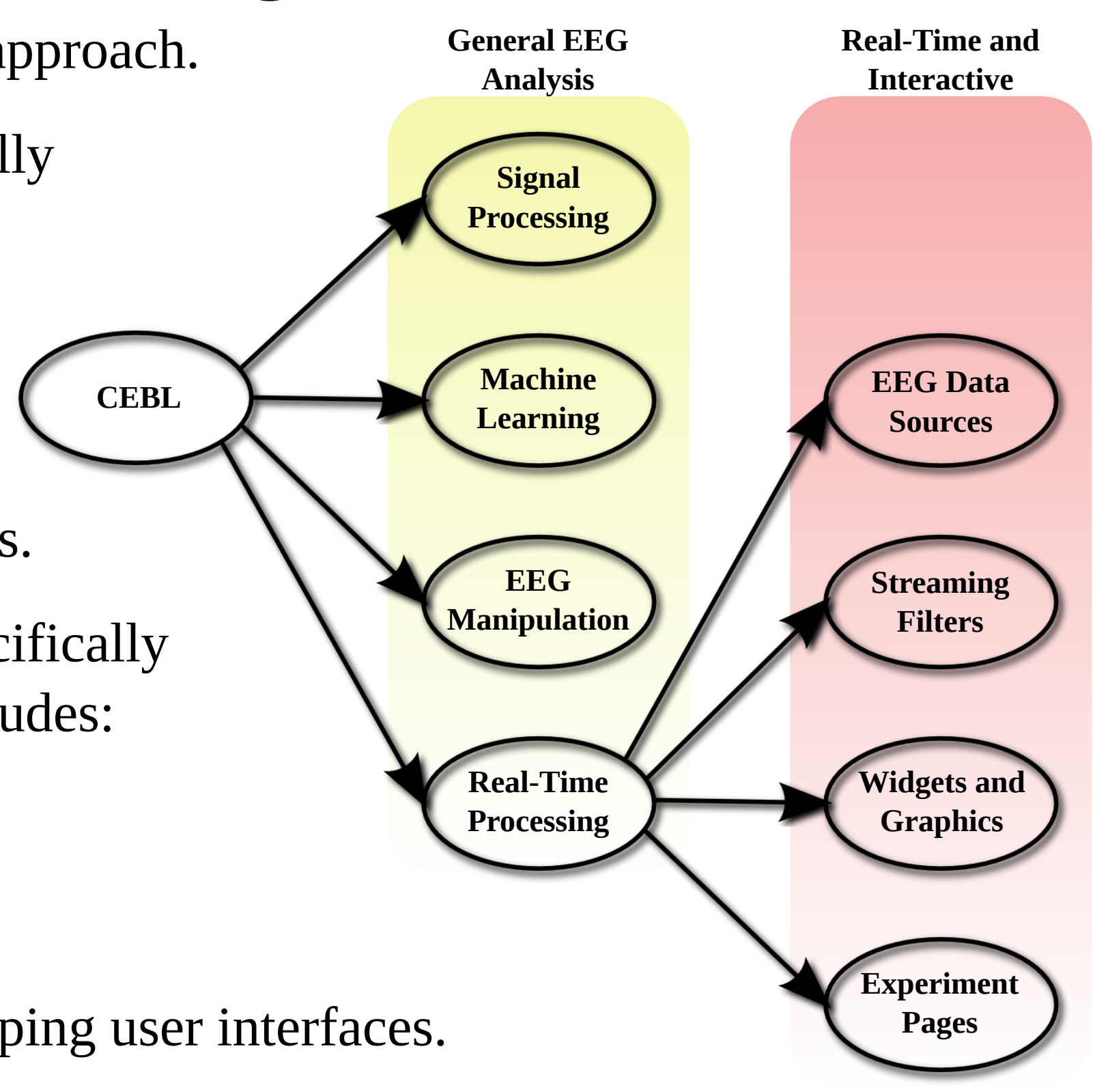
IP[y]: Notebook CEBL Notebook Demo Last Checkpoint: Dec 08 18:47 (autosaved)

```
In [14]: from cebl import *
         %matplotlib inline

         import warnings
         warnings.simplefilter('ignore')

In [19]: chanNames = ('F3','F4','C3','C4','P3','P4','O1','O2')
         s = eeg.EEGFromPickledMatrix(fileName='s11-letter-b.pkl.xz', sampRate=256, markers=8,
                                      chanNames=chanNames, transpose=True)

In [20]: s = s.demean().bandpass(0.5, 40.0, order=3)

In [21]: s.plotPSD(method='welch', span=5);

In [ ]:
```

-- INSERT --http://localhost:8888/notebooks/CEBL%20Notebook%20Demo.ipynb [-] [1/1]Top

Modular Design

CEBL$_3$ follows a modular, two-tiered design approach.



The first tier contains modules that are generally useful for EEG analysis. This includes:

-Signal processing routines.

-Machine learning algorithms.

-Classes for managing EEG segments and files.

The second tier contains modules that are specifically designed for real-time functionality. This includes:

-Drivers for EEG hardware.

-Filters for processing EEG data streams.

-Widgets and graphics components for developing user interfaces.

-Base-classes for developing experiments, called "pages."

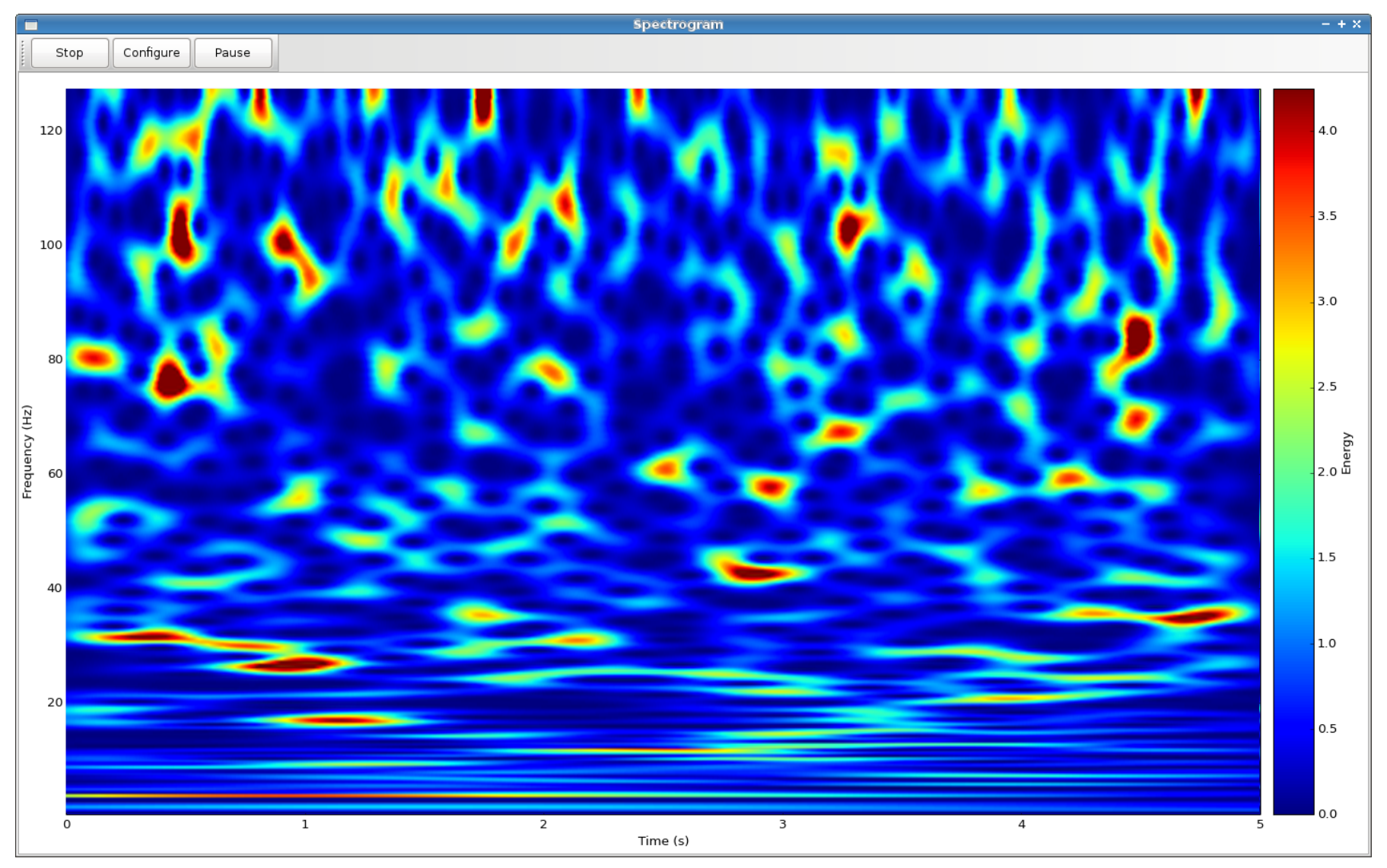
This two-tiered design is intended to follow the development process from beginning to end.

A researcher can first test new ideas in an offline fashion in the first tier. This can be done from an interactive environments like IPython.

Experiments that prove to be fruitful, can then be extended into real-time "pages" using the functionality of the second tier. This saves time by allowing the reuse of existing code.

Signal Processing

CEBL$_3$ currently includes a number of commonly used signal processing algorithms, such as:



-IIR and FIR linear filters

-Moving Average, Savitzky-Golay and Wiener filters

-Spectral estimation using Discrete Fourier Transforms, Welch's Method and Autoregressive Models

-Continuous Wavelet Transforms

A number of novel and advanced methods are also under development, including:

-Maximum Signal Fraction Filters

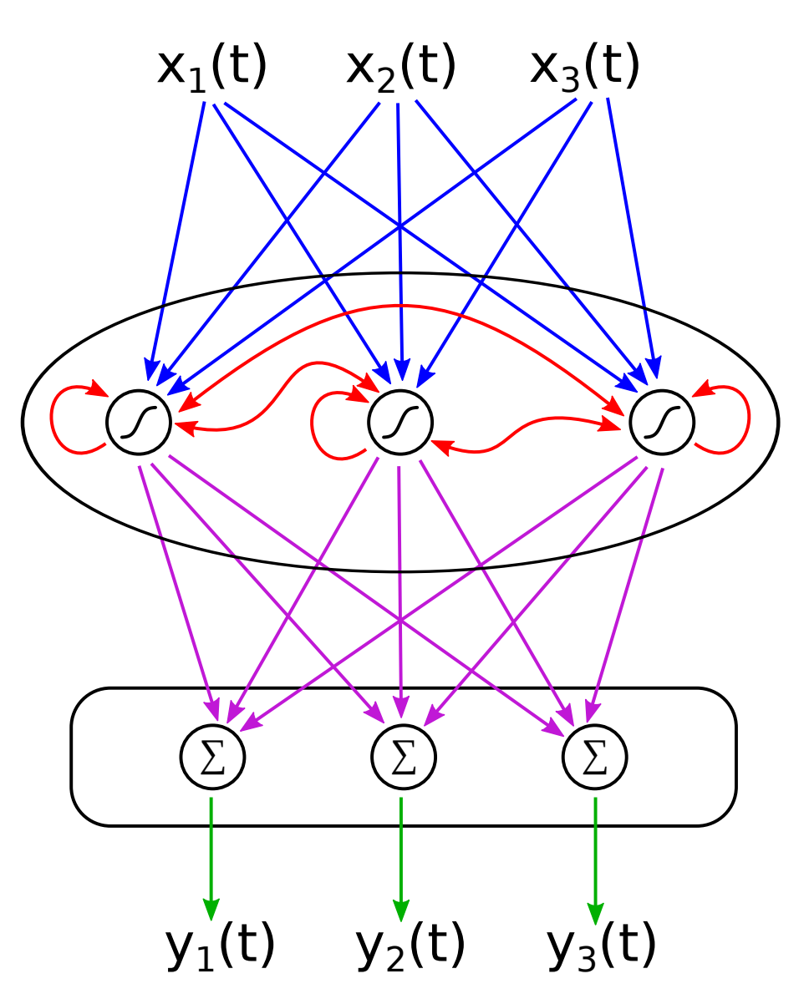
-Phase Locking Values

-Principal Components Analysis

-Smoothing by Regularization

Machine Learning

CEBL$_3$ currently includes a number of machine learning algorithms that are potentially useful for signal classification in Brain-Computer Interfaces, such as:

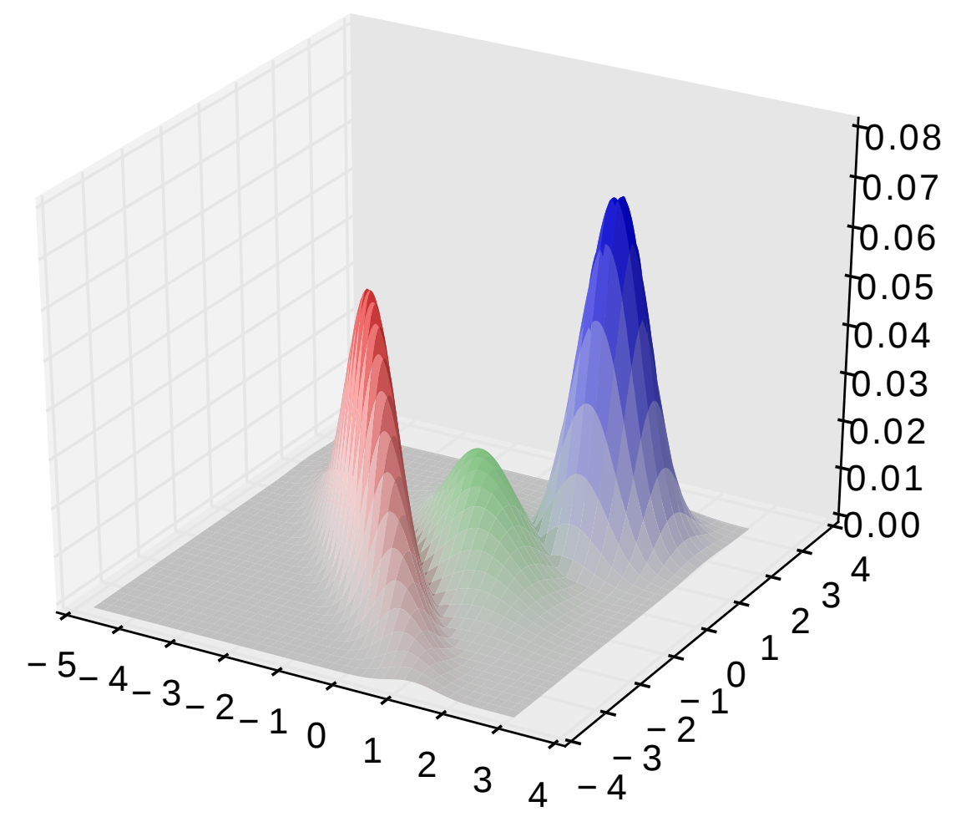


-Linear and Quadratic Discriminant Analysis

-Linear Logistic Regression

-Feedforward Artificial Neural Networks

-Linear and non-linear Autoregressive Models



A number of novel and advanced methods are also under development:

-Self-Organizing Maps

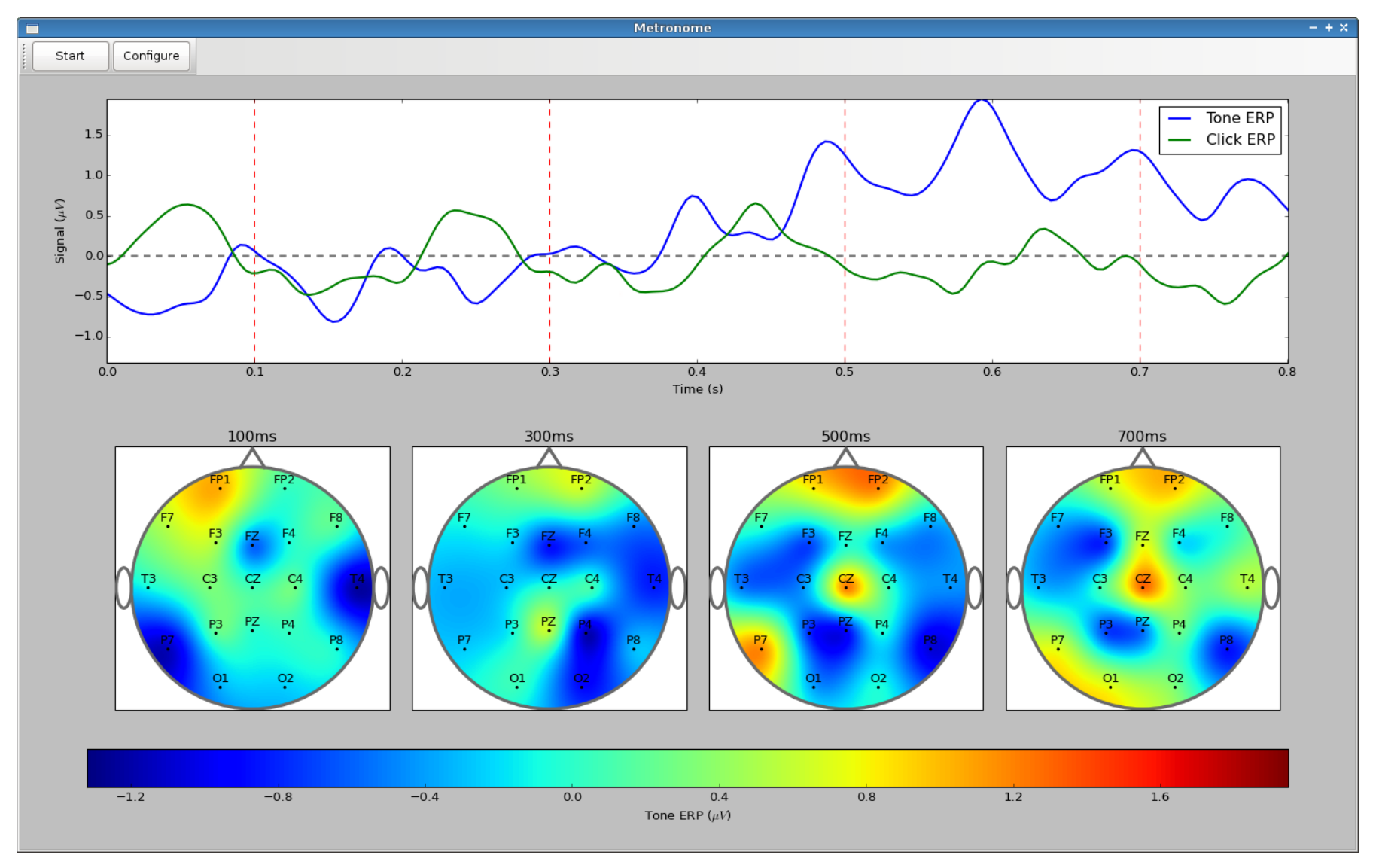
-Recurrent Artificial Neural Networks

-Support Vector Machines

-Kernel Logistic Regression

Visualization and Graphics

New ways of visualizeing EEG can easily be created in CEBL$_3$ simply by using the wxPython backend to the Matplotlib plotting library.

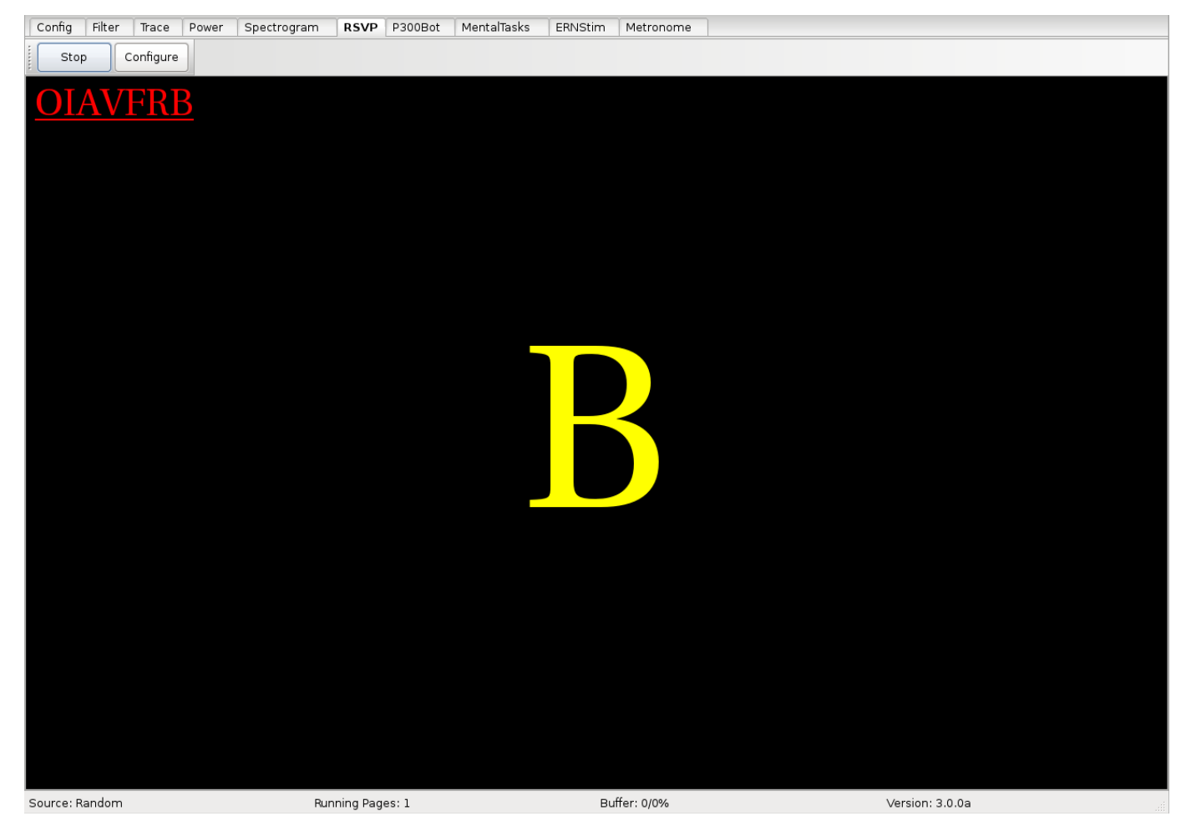


For fine-grained control or performance-intensive animated graphics, the standard wxPython graphics routines work well.

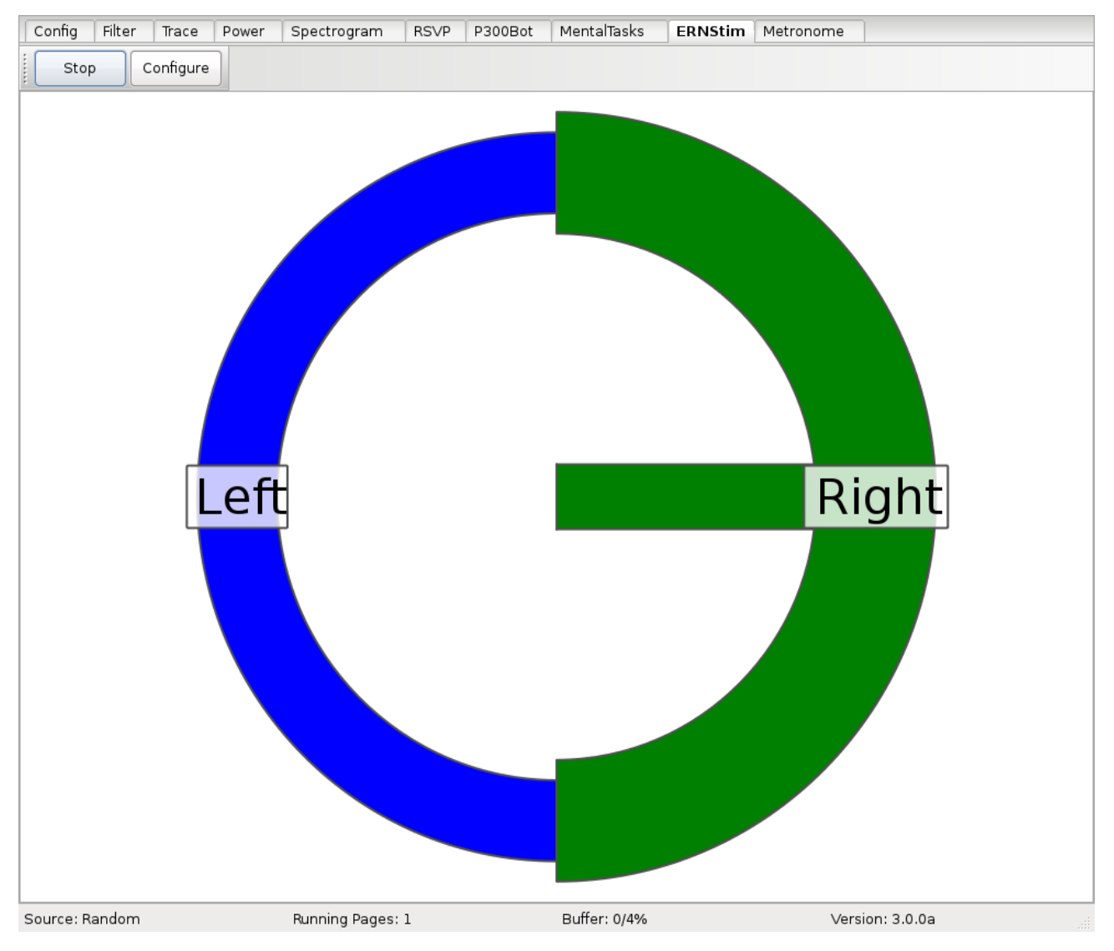
A number of pre-built widgets make it relatively simple for a researcher using CEBL$_3$ to reuse or extend various existing graphical components.

Interfaces

Construction of several prototype user-interface modules for CEBL$_3$ are currently underway.

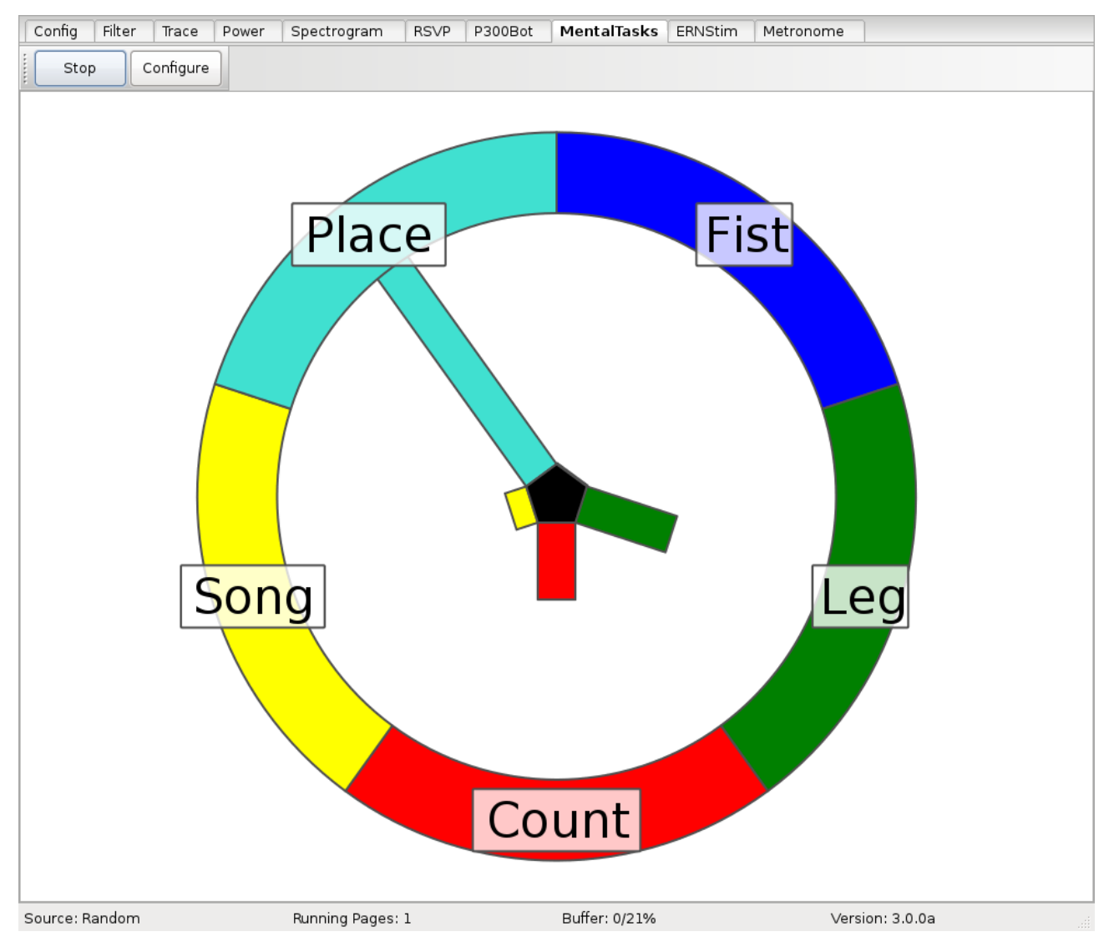


The first interface implements a P300 based BCI known as the Rapid Serial Visual Presentation (RSVP) speller. In this interface, the user is presented with a series of rapidly flashing characters. The user looks for the character they wish to spell while a machine learning algorithm attempts to identify changes in EEG when the correct character flashes.



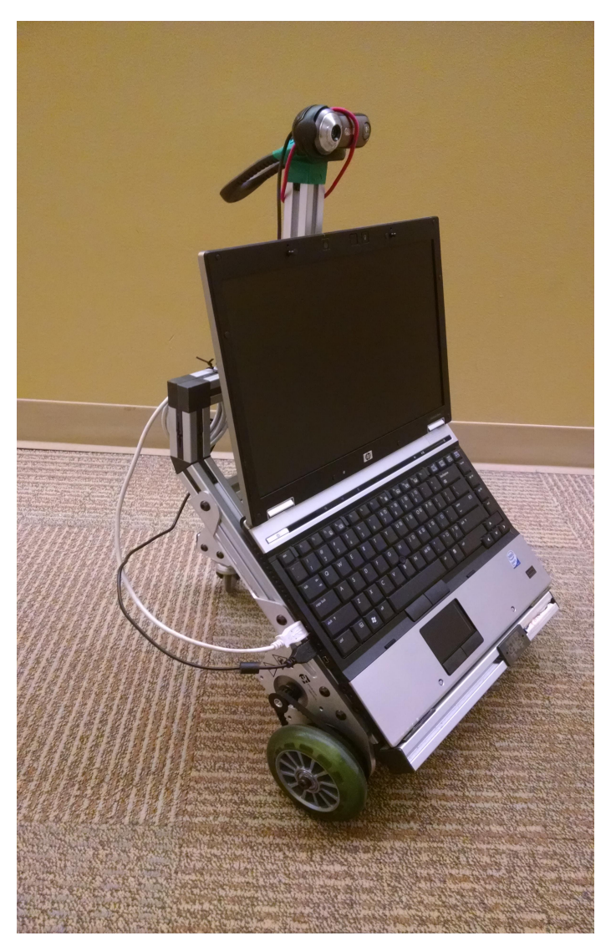
Another user-interface utilizes Motor Imagery (MI) to select one of two items from a circular "pie" menu. In this paradigm, the user simply imagines moving their left or right arm while a machine learning algorithm looks for changes in power in the mu band in the contralateral motor region of the brain.

The MI paradigm may also be combined with an Event-Related Potential known as Error-Related Negativity to identify when the BCI has made an incorrect selection.



A third interface is similar to the MI paradigm except that it allows a variety of imagined mental tasks that may elicit changes in activity over various regions of the brain. With practice, this paradigm may yield multiple degrees of freedom and fluid, asynchronous control.

Controlling Real-World Devices



The ultimate goal for BCI researchers is to construct systems that are practically useful.

To this end, we have implemented a module for driving an ER1 robot over a wireless connection.

We have also begun to implement several games that utilize box2d to create realistic simulations of physical environments.