mindaffectBCI

Release -

Jul 24, 2023

Setting up

1 What makes mindaffectBCI different **3**

2 Target users of mindaffectBCI **5**
2.1 User Interface Designers . . . 5
2.2 Neuroscience Students and Researchers . . . 5
2.3 Machine Learning Engineers / Data Scientists . . . 6

3 Project ideas for users **7**
3.1 Installation . . . 7
3.2 Supported EEG hardware . . . 12
3.3 Running on a Raspberry Pi . . . 14
3.4 Amplifier Setup: openBCI Cyton and Ganglion . . . 15
3.5 Quickstart . . . 18
3.6 How an Evoked Response BCI works . . . 24
3.7 Tutorials . . . 26
3.8 Going Further : BCI-types, Decoder Config . . . 65
3.9 Project Ideas / Inspiration . . . 67
3.10 Frequently Asked Questions . . . 67
3.11 Python API . . . 69
3.12 mindaffectBCI : System Overview and Component Roles . . . 159
3.13 System Overview . . . 159
3.14 mindaffectBCI : Message Specification . . . 167
3.15 Print Your Own Headset . . . 172
3.16 Set-up & fitting of MindAffect **water** electrodes . . . 173
3.17 Headset layout . . . 175

4 Indices and tables **179**

Python Module Index **181**

Index **183**

The mindaffectBCI is an open-source brain-computer-interface framework, aimed at enabling users to easily develop new ways for interacting with their computers directly using their brains.

CHAPTER 1

What makes mindaffectBCI different

There are already a number excellent of open-source BCI frameworks out there, such as bci2000 and openVibe, with large active user communities. So why develop another one?

When we looked at the existing frameworks, we noticed that whilst they were great and extremely flexible, they weren't so easy to use for developing end-user applications. So they didn't quite meet our objective of making it easy to develop new modes of interaction using brain signals.

Our aim is to simplify or hide the brain-signals aspect of the BCI as much as possible (or wanted) and so allow developers to focus on their applications. With this in mind mindaffectBCI is designed to be;

- **modular**
- **cross-platform** – run on all major desktop OSs (talk to us if you are interested in mobile)
- **hardware neutral** – support (via brainflow) many existing amplifiers, with instructions to easily add more,
- **language neutral** – provide APIs and examples for; python, java, c/c++, c#, swift
- **batteries included** – out of the box include a high-performance BCI, and examples for using this with common app development frameworks; (python, unity, swift)

CHAPTER 2

Target users of mindaffectBCI

2.1 User Interface Designers

The main target users for the mindaffectBCI are Application Developers who would like to add brain control to their applications, for example to add brain controls to a smart TV application. Within that we provide tools for particular user groups.

- Game Designers: Do you want to add brain controls to an existing game? Or make a new game including Brain controls as a novel interaction modality? You can easily do this, in a cross-platform way, using our unity plugin available here.
- Patient Technical Support Teams: One of the key motivators behind the MindAffect team is to make BCIs available to improve peoples lives. We can help some patients directly ourselves, but cannot support every possible patient and their environment. Instead, we try to provide the tools so patient support teams can themselves fit the BCI to their patients needs. For this, we provide a basic text communication application out-of-the-box, with guidance on how to customise this for their users needs, for example for fewer or more letters, or control of novel output devices.
- Hackers and Makers: Do you want to add brain control to your raspberry-pi robot, Lego robot, sphero or drone? Now you can, either by using a simple control app on your laptop, or (more fun) by adding LEDs or LASERS(!!!) to your robot for direct control. We provide examples for driving LEDs from a raspberry Pi, and are happy to help using other hacker boards (micro:bit) or even the LEDs on your drone.

2.2 Neuroscience Students and Researchers

For the user interface designers, we deliberately hide the brain-signals as much as possible. However, a BCI also provides an excellent tool for learning about basic neuroscience – in particular how the brain responses to external stimulus. For these users provide tools for the on-line real-time visualization of the stimulus specific responses. Importantly, these visualizations utilize the same technology as the on-line BCI, which uses machine learning techniques to improve signal quality and separate the responses from overlapping brain responses. This, gives students a clear

view of the brain response in a short amount of time allowing for interactive learning and experimentation with stimulus parameters or mental strategies. For example, so students can directly see the common (p300) and differential (perceptual) responses when using visual vs. auditory oddball paradigms.

2.3 Machine Learning Engineers / Data Scientists

Modern BCIs (including our own) rely heavily on machine learning techniques to process the noisy data gathered from EEG sensors and cope with the high degree of variability in responses over different individuals and locations. MindAffect firmly believes that with more sophisticated machine learning techniques more useful information can be extracted from even 'low quality' consumer grade EEG data. What is really needed is a combination of more and larger datasets on which to train the algorithms and better techniques tuned to the specific issues of neural data. The mindaffect BCI aims to facilitate this data lead approach to BCI in two ways.

- Firstly, by making it easier to rapidly gather relatively large EEG datasets by using consumer grade EEG devices and applications designed in your prefered application development framework. For example, by using a raspberry Pi, headphones, and EEG headband and an openBCI ganglion to measure the brain's response to different music types.
- Secondly, by providing a sklearn compatiable interface for machine learning developers to experiment with different learning algorithms, both in larger off-line dataset analysis and then directly in on-line applications.

CHAPTER 3

Project ideas for users

1) Brain controlled robot arm - use a laser or projector to illuminate objects to move, e.g. chess pieces, or food to eat, and the BCI to select which piece to move and where too. Lazy chess or snacking. See here for an example.
2) Neural shazam or Perceived music detection - Identify what music someone is listening to directly from their brain response.
3) Tactile BCI - Allow someone to answer yes-no-questions (or even spell words) by concentrating on different parts of their body.
4) Brain Defenders – Play missile-command using only your brain to pick where to send your defending missiles. Or go further and do it in Virtual Reality
5) Brain home-automation - Use brain control to change the color of your lights, like Philips Hue control, or to control your TV.
6) Real-world telekinesis - Use your brain to shoot storm-troopers in a modern tin-can-alley, like this
7) Brain-Golf (or Croquet)– play golf with your brain by controlling a sphero from a tablet. See Sphero control for some inspiration.
8) Brain control of your phone? Use our unity or iOS APIs to build a phone app controllable with your brain?Like this

3.1 Installation

3.1.1 Requirements

Warning: Numpy version 1.19.4 does currently not work under Windows 10-2004. Installation gives the following error: `RuntimeError: The current Numpy installation ('..\lib\\site-packages\\numpy\\__init__.py') fails to pass a sanity check due to a bug in the windows runtime`

If you are running into this issue, install numpy 1.19.3 manually:

```
pip install numpy==1.19.3
```

- Python (3.x) installation (Suggested: Anaconda)
- JDK such as openJDK
- EEG amplifier (e.g. OpenBCI Cyton, OpenBCI Ganglion, see supported_hardware for the full list of supported hardware.)

3.1.2 Recommended Setup

The MindAffect BCI can be used with various EEG acquisition and presentation devices and is therefore not limited to our recommended setup. It is however tested on the following supported hardware:

- Amplifier: OpenBCI Cyton
- Headset: The MindAffect Headset (Find the files to 3d print your own on our Github)
- Windows 10 machine

3.1.3 OS Optimization

For rapid visual stimulation BCI (like the MindAffect BCI), it is very important that the visual flicker is displayed accurately. From our testing we found that the following things help in improving timing accuracy:

- For windows 10: Disable full-screen optimization for your Python executable as explained here.
- For laptop users: make sure that your charger is plugged in and your machine is in *power mode: Best performance* (or a similar). How to change power mode.
- Set you screen to maximum brightness and disable *Night Light*, *f.lux*, or other applications that change the colour temperature of your screen.

3.1.4 Installing the package

To install from source:

1. Clone or download the pymindaffectBCI repository:

   ```
   git clone https://github.com/mindaffect/pymindaffectBCI
   ```

2. Install the necessary bits to your local python path:

   1. change to the directory where you cloned the repository.
   2. Add this module to the python path, and install dependencies:

      ```
      pip install -e .
      ```

   Note: The "." after "-e" should be included in the command!

To install as a python library. (Note: installing from source is recommended as you can directly access the examples and configuration files.):

```
pip install --upgrade mindaffectBCI
```

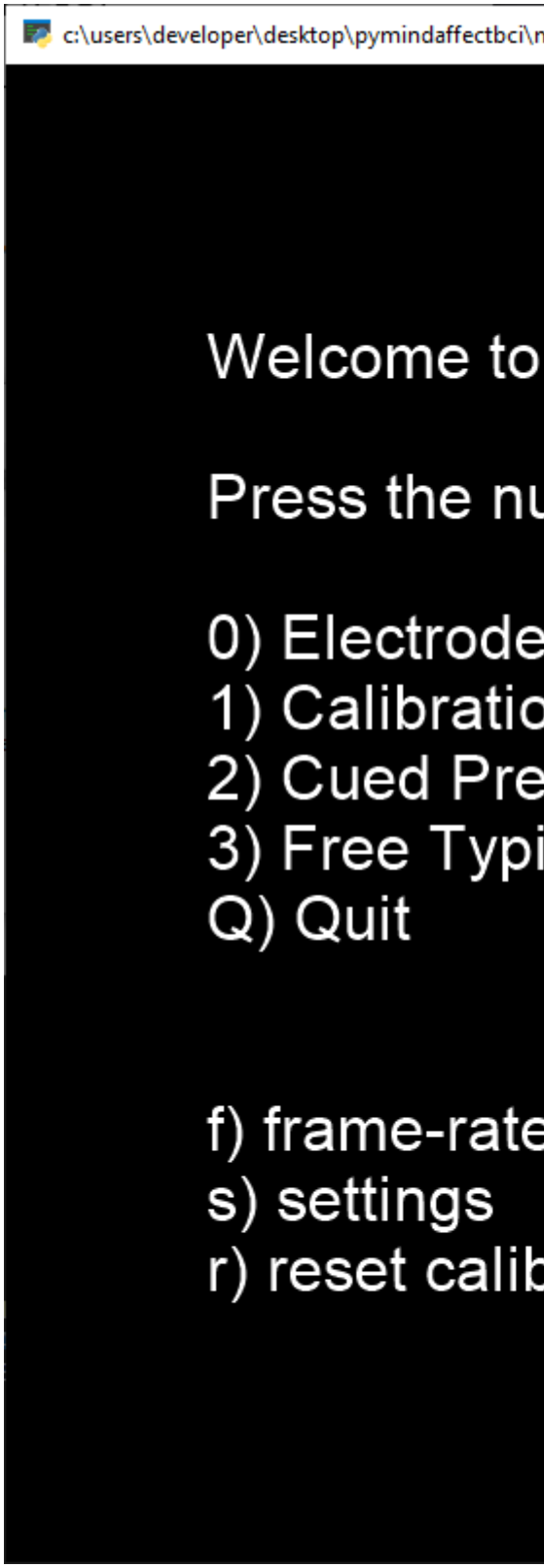
3.1.5 Installation Check

As a quick check if the software has installed correctly into your python environment you can run:

```
python3 -m mindaffectBCI.online_bci --acquisition fakedata
```

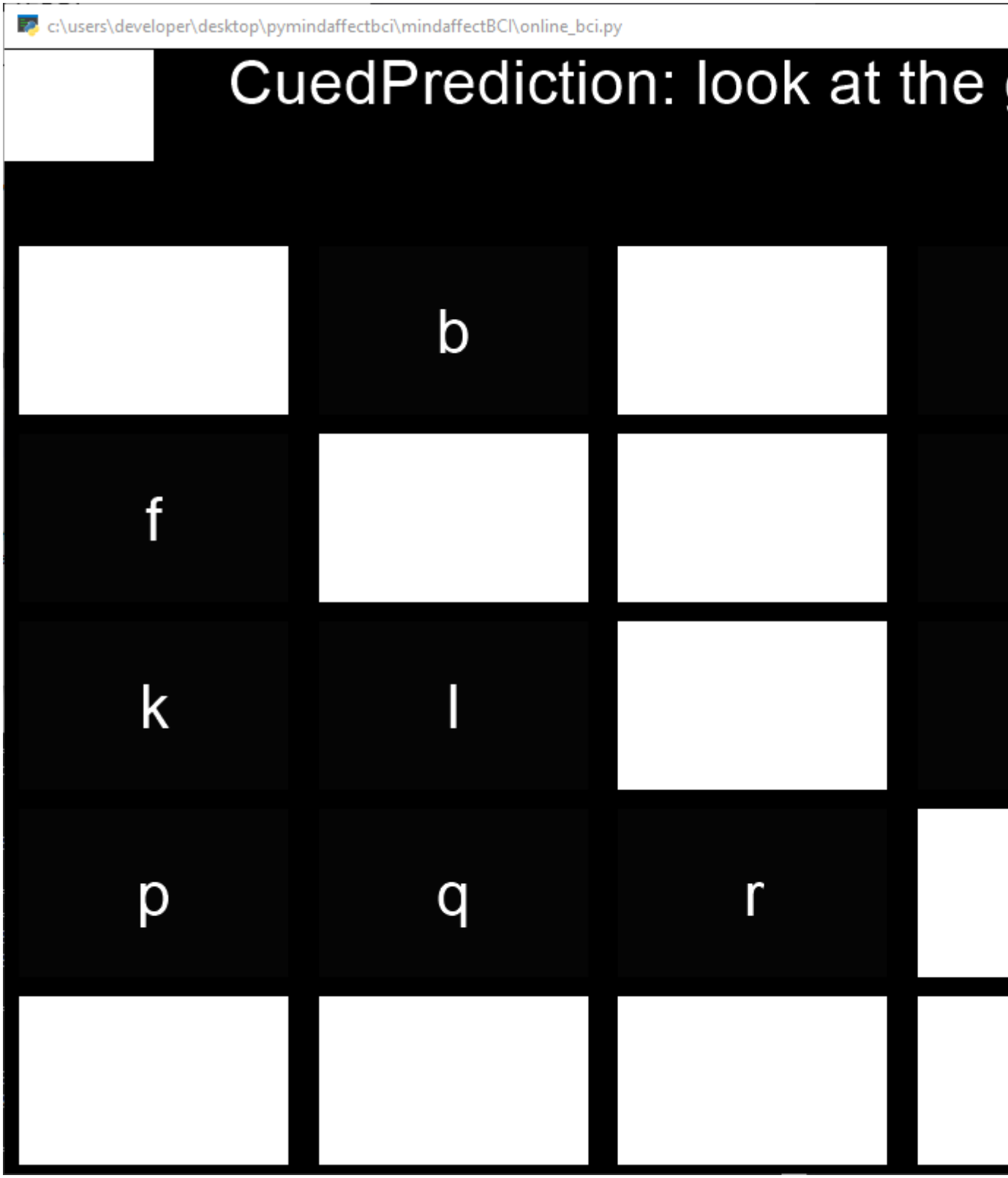
Note: depending on the specifics of your python installation, you may have to ommit the 3 from the command:

```
python -m mindaffectBCI.online_bci --acquisition fakedata
```



If all is successfully installed then you should see a window like this open up::

If you now press 2 you should see a flickering grid of "buttons" like below. You should see a random one briefly flash green (it's



If all this works then you have successfully installed the mindaffectBCI python software. You should now ensure your hardware (display, amplifier) is correctly configured before jumping into BCI control.

3.1.6 FrameRate Check

For rapid visual stimulation BCI (like the noisetagging BCI), it is very important that the visual flicker be displayed accurately. However, as the graphics performance of computers varies widely it is hard to know in advance if a particular configuration is accurate enough. To help with this we also provide a graphics performance checker, which will validate that your graphics system is correctly configured. You can run this with:

```
python3 -m mindaffectBCI.examples.presentation.framerate_check
```

or:

```
python -m mindaffectBCI.examples.presentation.framerate_check
```

As this runs it will show in a window your current graphics frame-rate and, more importantly, the variability in the frame times. For good BCI performance this jitter should be <1ms. If you see jitter greater than this you should probably adjust your graphics card settings. The most important setting to consider is to be sure that you have vsync turned-on. Many graphics cards turn this off by default, as it (in theory) gives higher frame rates for gaming. However, for our system, frame-rate is less important than exact timing, hence always turn vsync on for visual Brain-Compuber-Interfaces!

3.1.7 Amplifier configuration

In addition to configuring the software, you should ensure that your EEG hardware is correctly configured to optimise BCI performance. For the openBCI cyton and ganglion see this page *Amplifier Setup: openBCI Cyton and Ganglion* for how to configure these amplifiers and get the required COM-port information.

3.2 Supported EEG hardware

The mindaffectBCI can supports different ways to connect to amplifiers allowing us to work with many different amplifiers. Here we give an overview of the main connection types and the supported amps.

3.2.1 BrainFlow Connections

The primary amplifier connection type is provided by brainflow. Thus any amplifier supported by brainflow can be used by the mindaffect BCI with a simple configuration file change.

Internally, we have most experience with:

- openBCI Ganglion with 4 water-based EEG electrodes over the occiptial cortex using our 3d printed headband.
- openBCI Cyton with 6 to 8 water based EEG electrodes over the occiptial cortex using our 3d printed headband

3.2.2 Brainflow supported

This online_bci uses brainflow by default for interfacing with the EEG amplifier. Specificially the file in examples\acquisition\utopia_brainflow.py is used to setup the brainflow connection. You can check in this file to see what options are available to configure different amplifiers. In particular you should setup the *board_id* and and additional parameters as discussed in the brainflow documentation.

You can specify the configuration for your amplifer in the *acq_args* section of the configuration file online_bci.json. For example to specify to use the brainflow simulated board use

```
"acq_args":{ "board_id":-1}
```

Or to use the openBCI Cyton on com-port 4

```
"acq_args":{
    "board_id":0,
    "serial_port":"COM4"
 }
```

3.2.3 LSL supported

If your amplifier supports streaming with the Lab-Streaming-Layer then directly use this as an acquisition device. Specificially the file in examples\acquisition\utopia_lsl.py is used to setup the LSL connection. You can check in this file for detailed (and up-to-date) information on what options are available for the LSL connection.

You can specify the configuration for your amplifer in the *acq_args* section of the configuration file. For example to use connect to the 1st LSL device with the EEG datatype, and only stream channels named Cz,C3,C4 use:

```
"acquisition": "lsl",
"acq_args":{ "streamtype":"EEG", "channels":["Cz","C3","C4"] }
```

Note: This does *not* actually start the amplifier stream. You will need to separately do that by separately running the appropriate amplifier-lsl driver. For example using the openBCI_LSL drivers to configure and start an LSL stream for a Cyton.

3.2.4 FieldTrip Realtime supported

FieldTrip is a toolbox for analysis of neuroimaging data. As part of it's fieldtrip realtime support it provides drivers for a range of EEG/MEG/fMRI amplifiers. If your amplifier is supported in this way, you can use the FieldTrip acquisition driver to forward data from the fieldtrip amplifier driver to pymindaffectBCI.

You can specify the configuration for your amplifer in the *acq_args* section of the configuration file. For example to forward from FieldTrip buffer to pymimdaffectBCI use:

```
"acquisition": "ft"
```

Note: This does *not* actually start the amplifier stream. You will need to separately do that by separately running the appropriate amplifier-ft driver, see the list of implementations.

3.2.5 BrainProducts LiveAmp

Thanks to valuable support from BrainProducts, including loan equipment for testing, MindAffectBCI includes 'out-of-the-box' basic support for the BrainProducts LiveAmp. Specificially the file in examples\acquisition\utopia_brainproducts.py is used to connect to the amplifier. You can check in this file for detailed (and up-to-date) information on what options are available for this amplifier.

Note: To use this amplifier you must:

1. *first* install the amplifier driver, which you should have recieved along with your amplifier) *and*
2. have attached your 'key-dongle' to an available USB port on your computer.

You can specify the configuration for the LiveAmp in the *acq_args* section of the configuration file. For example to use this amp with default configuration use:

```
"acquisition": "bp"
```

3.2.6 AntNeuro eego

Thanks to valuable support from AntNeuro, including loan equipment for testing, MindAffectBCI includes 'out-of-the-box' basic support for the ANT-NEURO EEGO. Specificially the file in examples\acquisition\utopia_eego.py is used to connect to the amplifier. You can check in this file for detailed (and up-to-date) information on what options are available for this amplifier.

Note: To use this amplifier you must *first* install the amplifier driver, which you should have recieved along with your amplifier.

You can specify the configuration for the eego in the *acq_args* section of the configuration file. To use this driver with default config use

```
"acquisition": "eego"
```

3.2.7 Other Amplifiers

Alternatively, thanks to valuable support from their developers, we support some non-brainflow amplifiers 'out-of-the-box', spe

- TMSi Mobita: using *–acquisition mobita*, see examples\acquisition\utopia_mobita.py for the configuration options.

3.2.8 Add your own AMP support

If you have an amp which is not currently supported, and you have a way of getting raw samples out of it, then you can easily (7 lines of Python!) add support for your device as described in the Add a new Amplifier tutorial.

Hardware **Makers**: We are also happy to add support for additional amplifiers if EEG makers request it and are willing to provide open-source SDKs and test hardware.

3.3 Running on a Raspberry Pi

The mindaffectBCI can run directly on a raspberry pi. However, your milage may vary depending on which model of pi you have, we recommend;

- To run the complete BCI stack, with *screen-based* presentation, acquisition, decoder, etc., a pi model 4
- To run just the decoder or decoder with LED based presentation, a pi model 3 or higher.
- To run just LED presentation, then a pi zero

Depending on which part of the BCI you want to run, you will need to install some additional packages. For the full-system depl

- Java:
- BLAS:

In additon to enable openGL in the desktop you should enable the KMS (or fake-KMS) GL driver support by.

- open raspi-config
- Go to Advanced Options -> GL Driver -> GL (Fake KMS)
- reboot your Pi for the settings to take effect

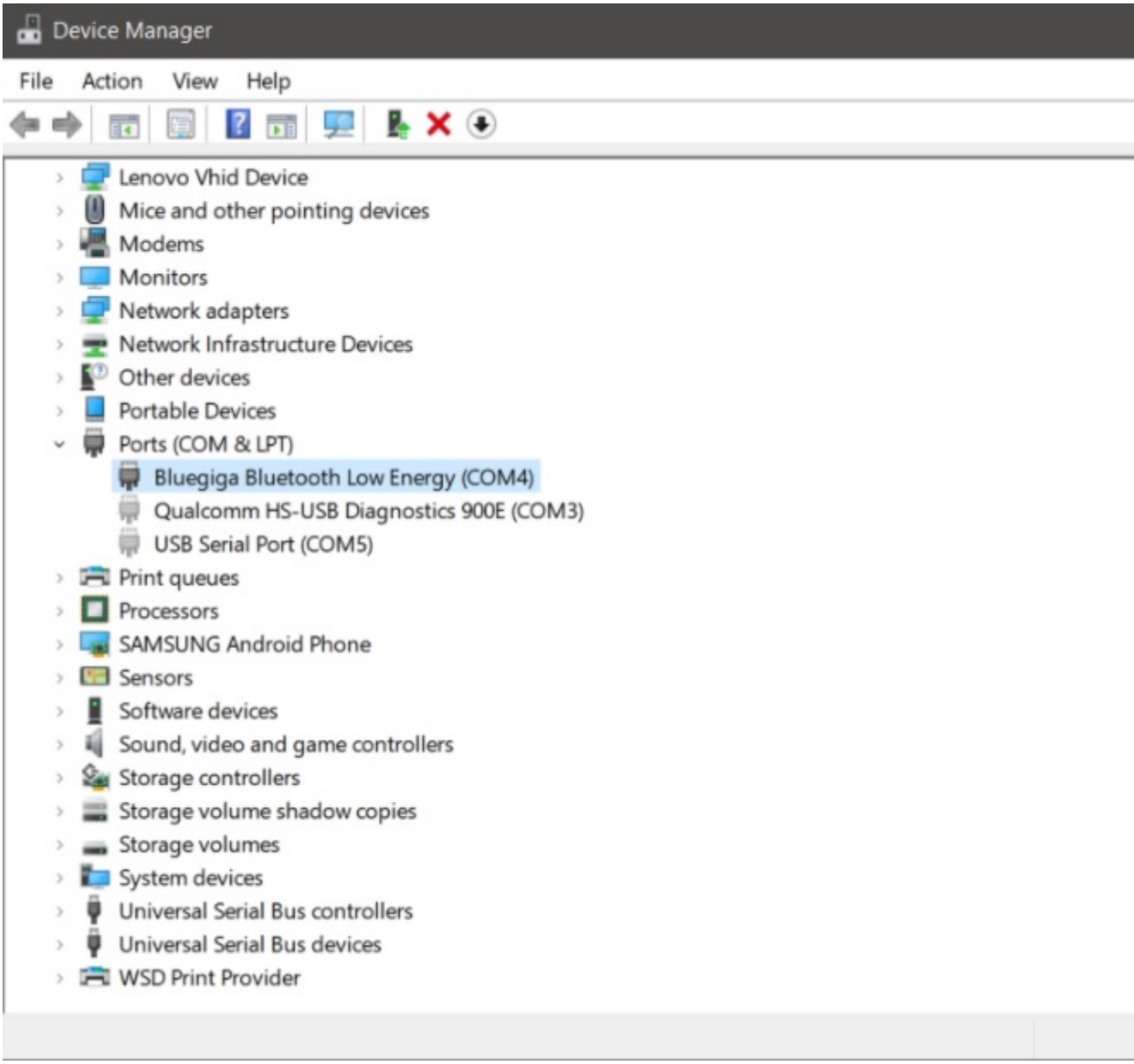
3.4 Amplifier Setup: openBCI Cyton and Ganglion

3.4.1 COM port

When using either the OpenBCI Ganglion or Cyton *with an USB-dongle* we have to pass the serial_port argument, to find the serial port in use by your amplifier follow the following instructions:

On Windows

1. Open Device Manager and unfold Ports(COM&LPT), the com port number is shown behind your used bluetooth adapter.



Then, in the online_bci configuration file (mindaffectBCI/online_bci.json) you should have: `"serial_port": "COM_X_"`. Also make sure to set `"board_id":0` to the value that corresponds with your amplifier as specified in the BrainFlow Docs.

On Mac

1. Open a Terminal session
2. Type: `ls /dev/cu.*`, and look for something like `/dev/cu.usbmodem1` (or similar):

```
            $ ls /dev/cu.*
/dev/cu.Bluetooth-Modem                     /dev/cu.iPhone-WirelessiAP
/dev/cu.Bluetooth-PDA-Sync   /dev/cu.usbserial
/dev/cu.usbmodem1
```

Then, in the online_bci configuration file (mindaffectBCI/config/online_bci.json) you should define as `"serial_port":"dev/cu.your_com_name"`. Also make sure to set `"board_id":0` to the value that corresponds with your amplifier as specified in the BrainFlow Docs.

On Linux

1. Open a Terminal session
2. Plug in your USB dongle
3. Type `dmesg`, and look for something like `ttyACM0` or `ttyUSB0` (or similar):

```
[43.364199] usb 2-1: New USB device found, idVendor=2458, idProduct=0001,␣
↪bcdDevice= 0.01
[43.364206] usb 2-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[43.364209] usb 2-1: Product: Low Energy Dongle
[43.364213] usb 2-1: Manufacturer: Bluegiga
[43.364215] usb 2-1: SerialNumber: 1
[43.394168] cdc_acm 2-1:1.0: ttyACM0: USB ACM device
```

Then, in the online_bci configuration file (mindaffectBCI/online_bci.json) you should define as `"serial_port": "dev/ttyXXXX"`. Also make sure to set `"board_id":0` to the value that corresponds with your amplifier as specified in the BrainFlow Docs.

Linux Serial Port Permissions

As explained in the OpenBCI Docs, on Linux you need to have permission to access the serial ports of your machine. Otherwise, you will get the error Failed to connect using `/dev/ttyUSB0` or similar. To fix this follow their instructions:

1. First, verify if the user does belong to the *dialout* group using the `id` command.
   - Type `id -Gn <username>` in terminal and check if it prints dialout as one of the options.
   - Replace with your Linux username. Example: `id -Gn susieQ`
2. Next, add the user to the *dialout* supplementary group.
   - Type `sudo usermod -a -G dialout <username>` in terminal.
   - Example: `sudo usermod -a -G dialout susieQ`
3. Restart Ubuntu
4. Try `id` command again
   - Repeat step one

3.4.2 OpenBCI Cyton Latency Fix

If you are using the OpenBCI Cyton with the included USB dongle, the default COM config has to be changed to fix latency issues. The default config for the dongle driver sends very big data-packets relatively slowly. The fix is pretty simple, just drop the packet size. To do so:

1. Open device-manager
2. Find the dongle driver under the ports dropdown
3. Go to properties for this com port

4. Go to port-settings
5. Select Advanced
6. Reduce the receive buffer to 1024 Bytes
7. Reduce the latency timer to 6ms
8. Apply and reboot

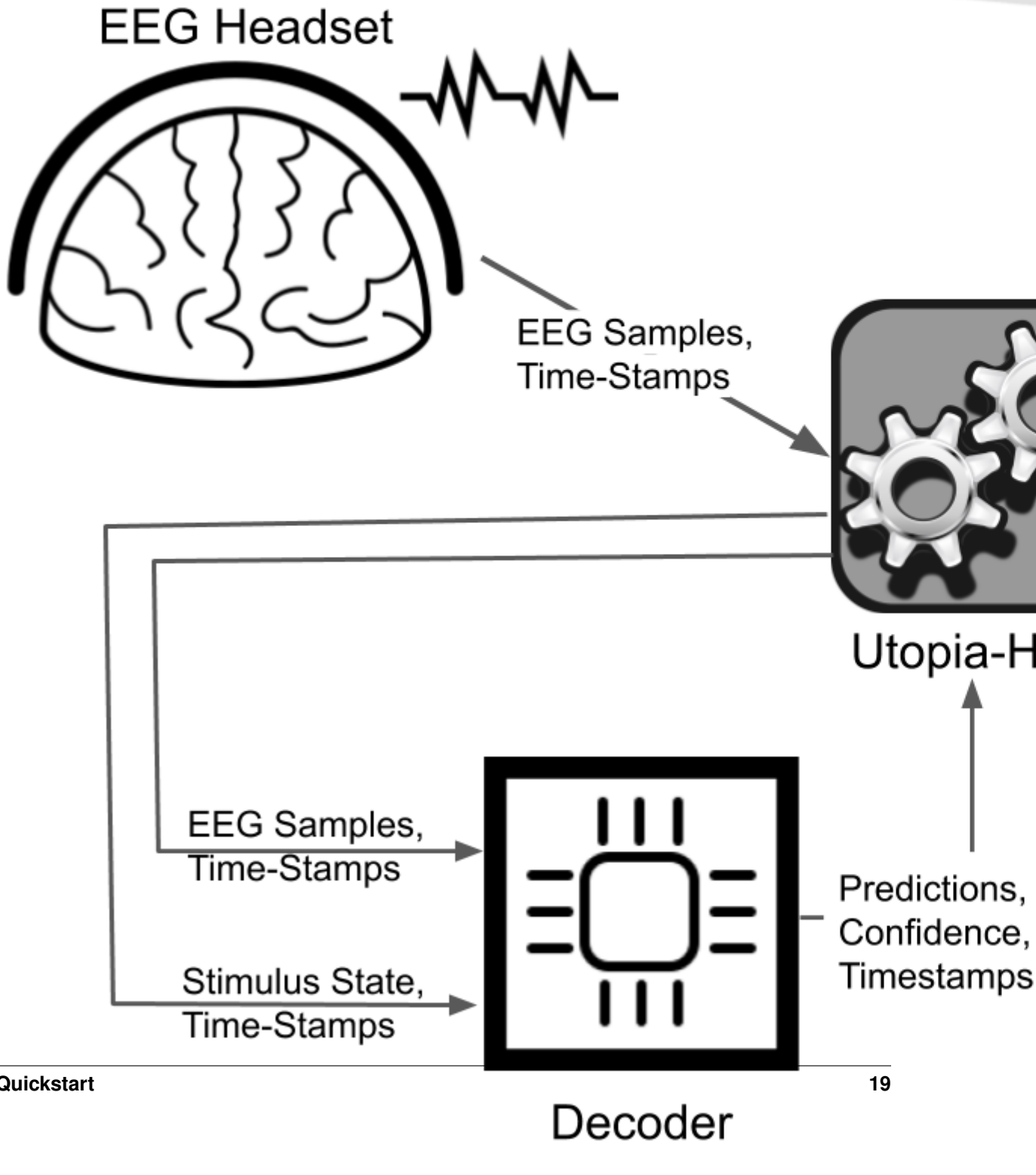
3.5 Quickstart

```
[ ]: %matplotlib qt
import mindaffectBCI.online_bci
```

3.5.1 System Architecture

The system consists of 3 main components as illustrated here.

mindaffectB
System Architecture - Roles



To actually run the BCI we need to start each of these components: - UtopiaHub: This component is the central server which coordinates all the other pieces, and saves the data for offline analysis

- Acquisition: This component talks to the *EEG Headset* and streams the data to the Hub
- Decoder: This component analysis the EEG data to fit the subject specific model and generate predictions
- Presentation: This component presents the User-Interface to the user, including any BCI specific stimuli which need to be presented. It also selects outputs when the BCI is sufficiently confident and generates the appropriate output

Before we launch the BCI to start these components: * Power on the OpenBCI Ganglion. (toggle on/off button)

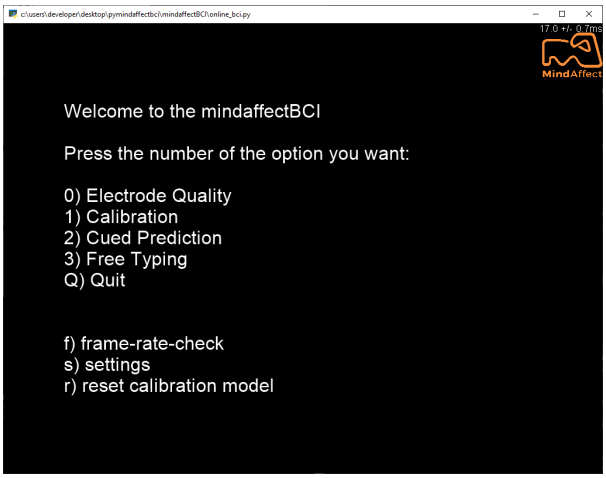
Now, we load the configuration we would like to use from the configuration .json file.

```
[ ]: # load the config-file to use.  Here we use the default noisetag config file:
config = mindaffectBCI.online_bci.load_config("noisetag_bci")
```

Then we can run this configuration, with the following command.

```
[ ]: # N.B. uncomment the following to run with fakedata driver, if you don't have an amp␣
     ↪connected
     # config['acquisition']='fakedata'

     mindaffectBCI.online_bci.run(**config)
```

If this worked correctly you should see a screen like below. (Note: it may start minimized to check for a new python window in your task bar).



3.5.2 Using the MindAffect BCI

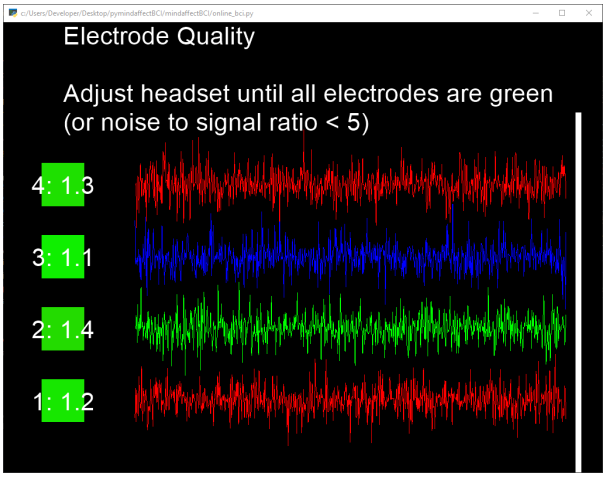
Now that the system is up and running, you can go through the following steps to use the BCI!

3.5.3 1. EEG headset Setup

Prepare a headset such that it follows the MindAffect headset layout.pdf in our Headset repository or prepare the headset delivered with your kit by following MindAffect headset setup.pdf

3.5.4 2. Signal Quality

Check the signal quality by pressing 0 in the main menu. You should see a screen which looks like this:



In this window you see a horizontial line for each channel showing the current *live* measurments. You also see a colored box at the left of each line with a number in it. The color of this box and the number indicate the current *estimated* noise-to-signal ratio for that channel. Ideally this number should be less than 5 and the box should be green.

It is critical for eventual BCI performance that the noise-to-signal ratio is as low as possible, and ideally less than 5.

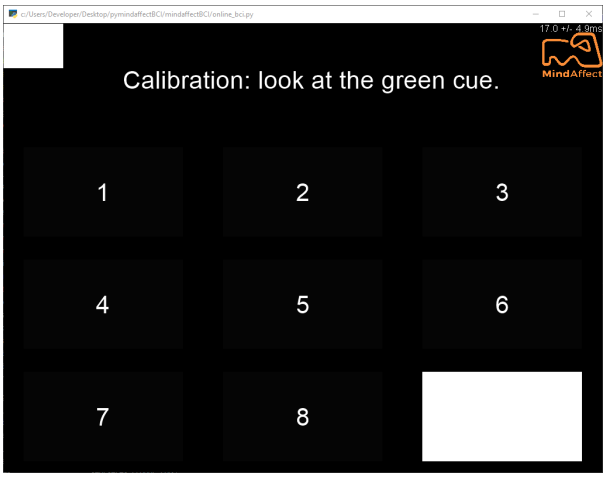
Try to adjust the headset until all electrodes are green, or noise to signal ratio is below 5.

You can try to improve the signal for an electrode by pressing it firmly into your head. After releasing pressure, wait a few seconds to see if the signal improves. If not, remove the electrode, and apply more water to the sponge. The sponges should feel wet on your scalp.

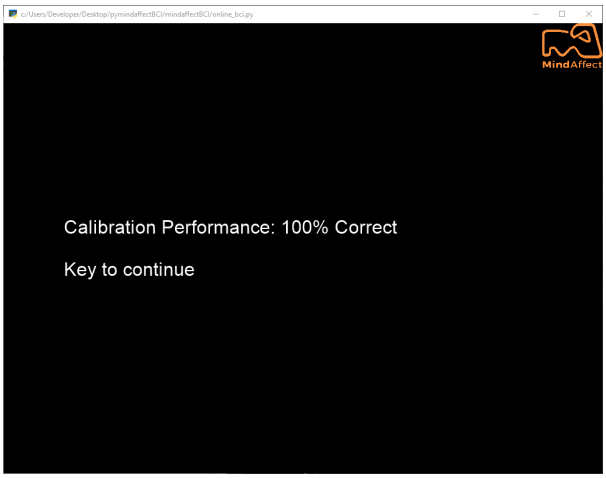
If the noise to signal ratio does not improve by adjusting the headset, try to distance yourself from power outlets and other electronics.

3.5.5 3. Calibration

Start calibration by pressing 1 in the main menu. You should see an instruction screen telling you to look at the green cued button. Followed by a screen which looks like this:



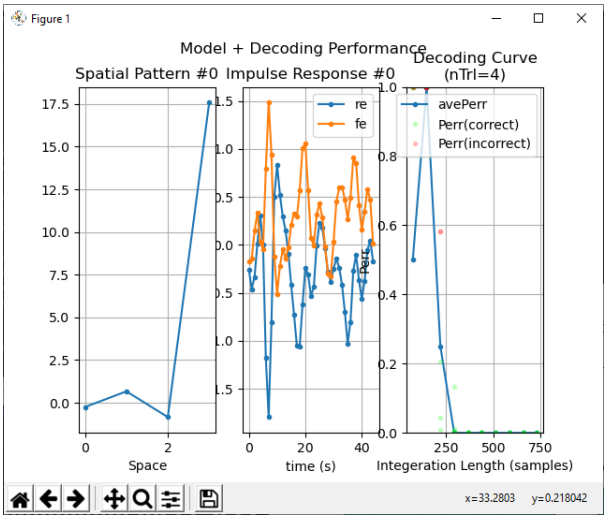
Do what it says, after 10 cues the stimulus will stop and after a few seconds you should see a 'calibration performance' screen like this:



Hopefully, you got a similar performance level?

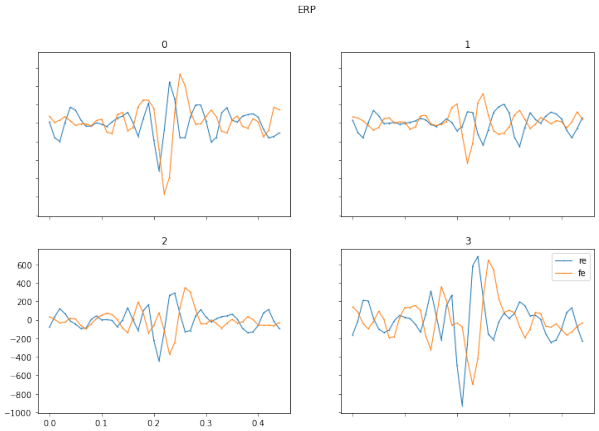
In addition you should see 3-windows pop-up showing more information about the performance. These windows are:

1. Model and Performance:



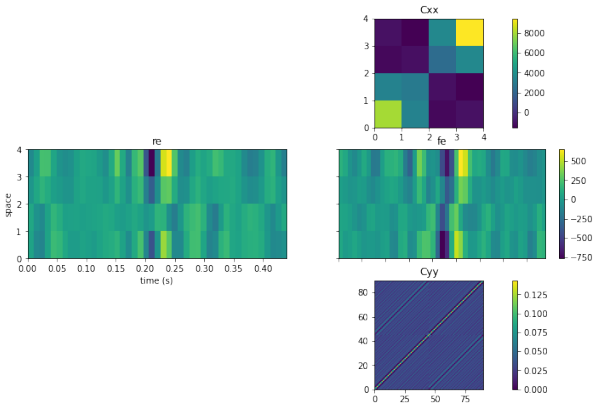
This window shows (in left-to-right order). a) the fitted models spatial-filter – which shows the importance of each EEG channel, b) the models impulse response – which shows how the brain responds over time to the different types of stimulus event, and c) the decoding-performance – which shows how the accurately the model is able to decode the cued target with increasing data length, and the models estimate of it's own accuracy.

2. ERP (Event Related Potentional):



This window shows for each EEG channel the *averaged* measured response over time after the triggering stimulus. This is the conventional plot that you find in many neuroscientific publications.

3. Summary Statistics:



This window shows the summary statistics for the calibration data. This has vertically 3 sub-parts. a) Cxx : this is the spatial cross-correlation of the EEG channels. b) Cxy : this is the cross-correlation of the stimulus with the EEG. Which for discrete stimuli as used in this BCI is essentially another view of the ERP. c) Cyy : this is the *temporal* cross-correlation of the stimulus.

3.5.6 4. Brain Control.

If calibration worked well and you have good calibration performance, then you can proceed to brain-control. Press either 3 for *cued* prediction, where like calibration you are told where to look. Whilst not much fun, this is useful to get on-line datasets for training improved models. Or press 4 for free-spelling, where you can select what 'button' you want in an un-cued fashion.

In both cases, BCI feedback is shown in blue. Initially, while the system is unsure by changing the color of the central letter, and later when a selection has been made by making the whole button blue. Selected letters will be added to the spelling box at the top of the screen.

Struggling to get the system to work? Consult our FAQ section for info on how to improve calibration accuracy, prediction performance, and more!

Do you not want to run this notebook everytime when using the BCI?

Simply run it from your command prompt:

```
python3 -m mindaffectBCI.online_bci
```

Or specify your own configuration file and save location:

```
python3 -m mindaffectBCI.online_bci --config_file noisetag_bci --logdir .
```

3.5.7 5. SHUTDOWN

In the mainmenu, press 5 to shutdown the BCI.

[ ]:
```
# shutdown the background processes.
# N.B. only needed if shutdown from the main-menu window doesn't work.
mindaffectBCI.online_bci.shutdown()
```

3.5.8 Going Further

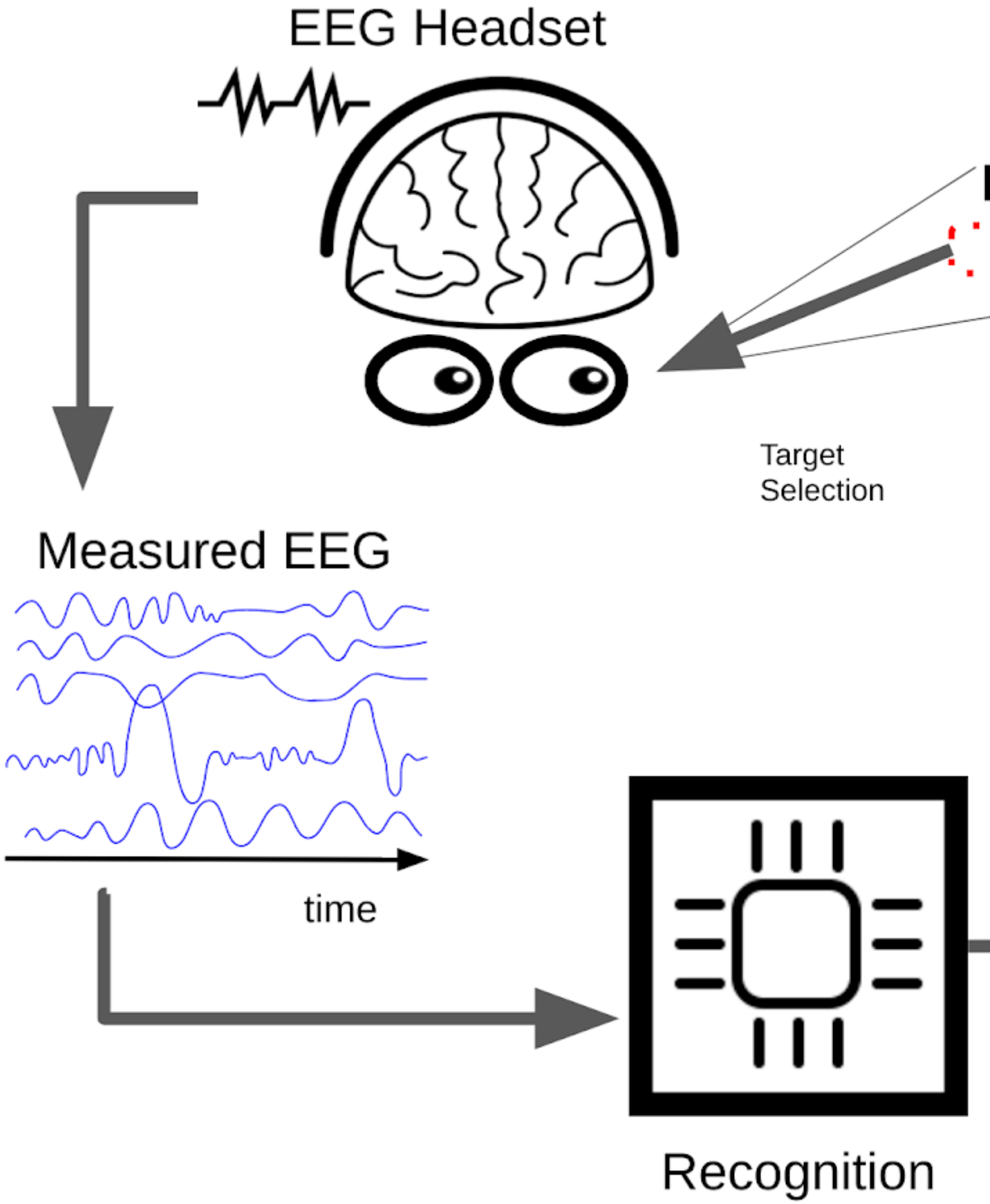
1. Try other BCI types using our alternative configuration files.
   - noisetag.json : example for a noise-tagging (or c-VEP) BCI (Default)
   - rc5x5.json : example for a classic visual P300 odd-ball type BCI with row-column stimulus.
   - ssvep.json : example for a classic steady-state-visual-response BCI.
2. Write your own presentation system by following this guide (https://mindaffect-bci.readthedocs.io/en/latest/simple_presentation_tutorial.html)
3. Write your own output system to make *interesting* things happen when a brain control is activated following this guide (https://mindaffect-bci.readthedocs.io/en/latest/simple_output_tutorial.html)
4. Build your own BCI following this guide to develop your own compents. (https://mindaffect-bci.readthedocs.io/en/latest/first_run.html)

[ ]:
```

```

3.6 How an Evoked Response BCI works

Evoked Response Brain Computer Interfaces, which are also called or Event Related Potential (ERP) BCIs, are a general class of Brain Computer Interface which rely on detecting the brain response to **known** external stimuluation. This type of BCI is one of the most successful BCI types for communication purposes, as many options can be presented in parallel and selected between – allowing one to for example implement a full keyboard for text generation. Further, whilst the most successfull ERP-BCIs have used discrete visual stimuli, the same basic approach can be used for other discrete stimlus types (such as auditory or tactile) or even continuous stimulus, such as natural speech https://doi.org/10.3389/fnins.2016.00349.

The schematic below illustrates the general principles of (visual) ERP BCIs:


EEG Headset
Target
Selection
Measured EEG
time
Recognition

To briefly describe this schematic, the aim of a BCI in general is to control an output device (in this case a robot arm) with your thoughts. In this specific case we control the robot arm by selecting actions perform as flickering virtual buttons on a tablet screen. (See Mindaffect LABS for a video of the system in action) :

1. Presentation: displays a set of options to the user, such as which square to pick in a tic-tac-toe game, or whether to pick something up with a robot arm.
2. Each of the options displayed to the user then flickers with a given unique flicker sequence (or stimulus-sequence) with different objects getting bright and dark at given times.
3. The user looks at the option they want to select to select that option. (Or in a non-visual BCI 'attends to' or 'concentrates on' the option they want to select.)
4. The users brain response to the presented stimulus is measured by EEG - due to the users focus on their target object, this EEG signal contains information on the flicker-sequence of the target object.
5. The recognition system uses the measured EEG and the known stimulus sequence to generate predictions for the probability of the different options being the users target option.
6. Selection takes the predictions from the recogniser and any prior knowledge of the application domain (such as a language model when spelling words) to decide when an option is sufficiently confidently predicted to be selected and output generated.
7. Finally, Output generates the desired output for the selected option, for example moving the robot arm to perform the selected option.

3.7 Tutorials

All of our tutorials are interactive Jupyter Notebooks. Clone our repository to run them locally from your *pymindaffectBCI/docs/source* directory:

```
jupyter notebook DIY_brain_computer_interfacing.ipynb
```

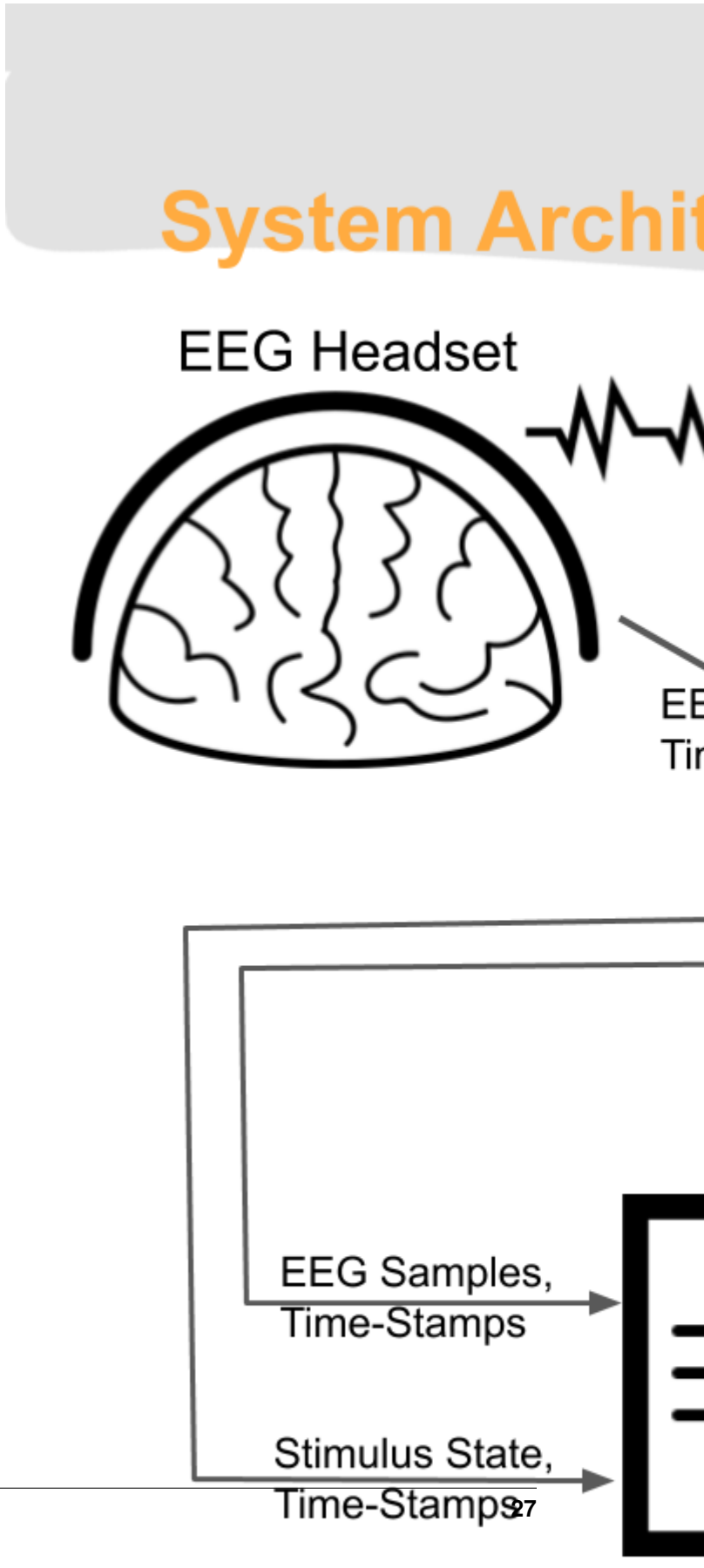
3.7.1 DIY Brain Computer Interfacing

This tutorial is a deeper dive into the mindaffectBCI, which shows you how to manually start each of the components individually. Before running this tutorial you should have read how an evoked bci works to get an overview of how this BCI works, and run through *quickstart tutorial* to quickly test your installation and try the BCI.

After following this tutorial, you will be able to; * configure the BCI startup to your needs, and * know how to replace any of the main components with your own version.

[ ]:
```
# Import the mindaffectBCI decoder and other required modules.
%load_ext autoreload
%autoreload 2
import mindaffectBCI.online_bci
```

System Architecture



To actually run the BCI we need to start each of these components: - UtopiaHub: This component is the central server which coordinates all the other pieces, and saves the data for offline analysis

- Acquisition: This component talks to the *EEG Headset* and streams the data to the Hub
- Decoder: This component analysis the EEG data to fit the subject specific model and generate predictions
- Presentation: This component presents the User-Interface to the user, including any BCI specific stimuli which need to be presented. It also selects outputs when the BCI is sufficiently confident and generates the appropriate output

First, we start the UtopiaHub by running the following code block:

[ ]:
```
#--------------------------- HUB -------------------------------
# start the utopia-hub process
hub_process = mindaffectBCI.online_bci.startHubProcess()
```

ACQUISITION

Now that the hub is running we want to establish a connection between the amplifier and the hub to stream the EEG data. To achieve this the Brainflow library is used. The brainflow driver has to be initialized with input parameters that depend on the amplifier in use: (Consult the Brainflow docs for a complete list of amplifiers supported by brainflow but currently untested with the MindAffect BCI.)

Board	`board_id`	`serial_port`	`ip_address`	`ip_port`
Ganglion	1	dongle serial port(COM3, /dev/ttyUSB0…)	•	•
Ganglion + WiFi Shield	4	•	WIFI Shield IP(default 192.168.4.1)	any local port which is free
Cyton	0	dongle serial port(COM3, /dev/ttyUSB0…)	•	•
Cyton + Wifi Shield	5	•	WIFI Shield IP(default 192.168.4.1)	any local port which is free

When using either the OpenBCI Ganglion or Cyton with an USB-dongle we have to pass the `serial_port` argument, to find the serial port in use by your amplifier follow the following instructions:

On Mac:

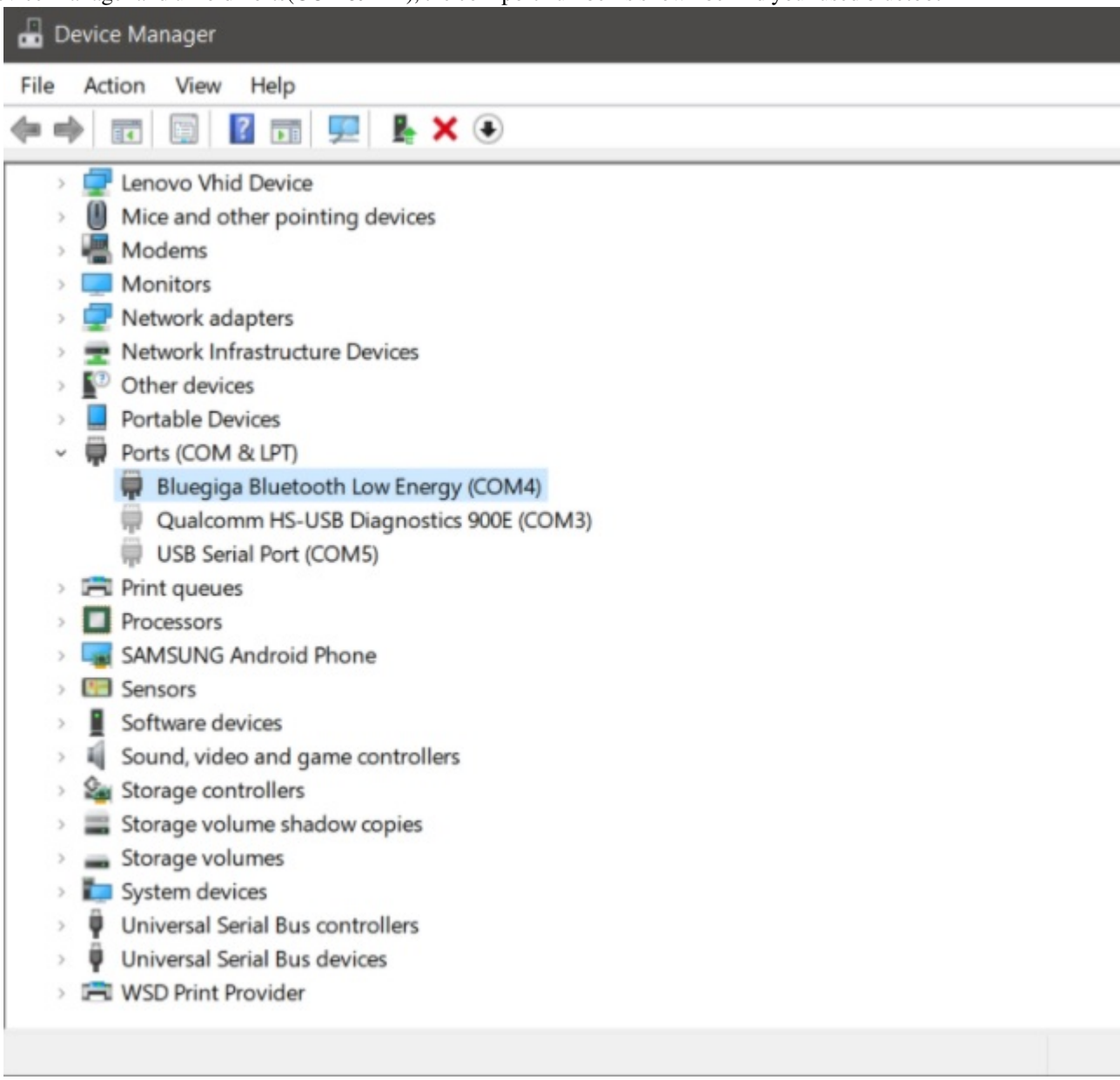
1. Open a Terminal session
2. Type: `ls /dev/cu.*`, and look for something like `/dev/cu.usbmodem1` (or similar):

   ```
   $ ls /dev/cu.*
   /dev/cu.Bluetooth-Modem     /dev/cu.iPhone-WirelessiAP
   /dev/cu.Bluetooth-PDA-Sync  /dev/cu.usbserial
   /dev/cu.usbmodem1
   ```

   Then, `serial_port` should be defined as `"serial_port":"dev/cu.your_com_name"`

On Windows:

1. Open Device Manager and unfold Ports(COM&LPT), the com port number is shown behind your used bluetooth adapter.



   Then, `serial_port` should be defined as `"serial_port":"COM_X_"`

The code below shows the `acq_args` for the Ganglion, change the arguments for the board in use and run the code block to start the acquisition process.

[ ]:
```
# start the ganglion acquisition process
# Using brainflow for the acquisition driver.
#  so change the board_id and other args to use other boards
acq_args =dict(board_id=1, serial_port='com4') # connect to the ganglion
acq_process = mindaffectBCI.online_bci.startacquisitionProcess('brainflow', acq_args)
```

N.B. Only use this cell if you just want to test with a *fake* eeg stream

```
[ ]: # start a fake-data stream
     # with 4-channels running at 200Hz
     acq_args=dict(host='localhost', nch=4, fs=200)
     acq_process = mindaffectBCI.online_bci.startacquisitionProcess('fakedata', acq_args)
```

DECODER

The decoder is the core of the BCI as it takes in the raw EEG and stimulus information and generates predictions about which stimulus the user is attending to. Generating these predictions relies on signal processing and machine learning techniques to learn the best decoding parameters for each user. However, ensuring best performance means the settings for the decoder should be appropriate for the particular BCI being used. The default decoder parameters are shown in the code below, and are setup for a noisetagging BCI.

```
[ ]: # start the decoder process, wih default args for noise-tagging
     decoder_args = dict(
             stopband=((45,65),(3,25,'bandpass')),  # frequency filter parameters
             out_fs=100,  # sample rate after pre-processing
             evtlabs=("re","fe"),  # use rising-edge and falling-edge as brain response␣
     ↪triggers
             tau_ms=450, # use 450ms as the brain response duration
             calplots=True, # make the end-of-calibration model plots
             predplots=False # don't make plots during prediction
         )
     decoder_process = mindaffectBCI.online_bci.startDecoderProcess('decoder', decoder_
     ↪args)
```

The key parameters here are:

- `stopband`: this is a temporal filter which is applied as a pre-processing step to the incoming data. This is important to remove external noise so the decoder can focus on the target brain signals.

Here the filter is specified as a list of band stop filters, which specify which signal frequencies should be suppressed, (where, in classic python fashion -1 indicates the max-possible frequency). Thus, in this example all frequencies below 3Hz and above 25Hz are removed.

- `out_fs`: this specifies the post-filtering sampling rate of the data. This reduces the amount of data which will be processed by the rest of the decoder. Thus, in this example after filtering the data is re-sampled to 80Hz. (Note: to avoid ‘ <>‘__ out_fs should be greater than 2x the maximum frequency passed by the stop-band).
- `evtlabs`: this specifies the stimulus properties (or event labels) the decoder will try to predict from the brain responses. The input to the decoder (and the brain) is the raw-stimulus intensity (i.e. it’s brightness, or loudness). However, depending on the task the user is performing, the brain may *not* respond directly to the brightness, but some other property of the stimulus.

For example, in the classic P300 ‘odd-ball’ BCI, the brain responds not to the raw intensity, but to the start of *surprising* stimuli. The design of the P300 matrix-speller BCI means this response happens when the user’s chosen output ‘flashes’, or gets bright. Thus, in the P300 BCI the brain responds to the rising-edge of the stimulus intensity.

Knowing exactly what stimulus property the brain is responding to is a well studied neuroscientific research question, with examples including, stimulus-onset (a.k.a. rising-edge, or ‘re’), stimulus-offset (a.k.a. falling-edge, or ‘fe’), stimulus intensity (‘flash’), stimulus-duration etc. Getting the right stimulus-coding is critical for BCI performance, see `` `stim2event.py <mindaffectBCI/decoder/stim2event.py>‘__ `` for more information on supported event types.

- `tau_ms`: this specifies the maximum duration of the expected brain response to a triggering event in *milliseconds*. As with the trigger type, the length of the brian response to a triggering event depends on the type of

response expected. For example for the P300 the response is between 300 and 600 ms after the trigger, whereas for a VEP the response is between 100 and 400 ms.

Ideally, the response window should be as small as possible, so the learning system only gets the brain response, and not a lot of non-response containing noise which could lead the machine learning component to overfitt.

PRESENTATION

Before launching the presentation component we first make sure that the Hub, Acquisition, and Decoder components are running:

```
[ ]: # check all is running?
print("Hub running {}".format(hub_process.poll() is None))
print("Acquisition running {}".format(acq_process.is_alive()))
print("Decoder running {}".format(decoder_process.is_alive()))
print("Everything running? {}".format(mindaffectBCI.online_bci.check_is_running(hub_
↪process,acq_process,decoder_process)))
```

If not, try running the corresponding codeblock of the inactive component before continuing.

Creating the Stimulus

By default we use the MindAffect NoiseTagging style stimulus with a 25-symbol letter matrix for presentation. You can easily try different types of stimulus and selection matrices by modifying the `symbols` and `stimfile`arguments. Where: * *symbols* : can either by a list-of-lists of the actual text to show, for example for a 2x2 grid of sentences:

```
[ ]:  symbols=[["I'm happy","I'm sad"], ["I want to play","I want to sleep"]],
```

or a file from which to load the set of symbols as a *comma-separated* list of strings like the file symbols.txt.

```
[ ]: symbols="symbols.txt"
```

- *stimfile* : is a file which contains the stimulus-code to display. This can either be a text-file with a matrix specified with a white-space separated line per output or a png with the stimulus with outputs in 'x' and time in 'y'. This is what the *codebook* for the noisetag looks like where symbols are left to right and time is top to bottom. 

Using Images as Stimuli

We also support the use of images as stimuli as shown by the robot_control.txt file. Simply specify the relative paths of the images you want to use as stimuli in a .txt file, or directly in a list-of-list as shown above.

Running the Stimulus

The UI with the desired presentation stimuli can be launched by running the code below. ### Note: the stimulus window my appear minimized, so check your task-bar!

```
[ ]: # run the presentation, with our matrix and default parameters for a noise tag
from mindaffectBCI.examples.presentation import selectionMatrix
selectionMatrix.run(symbols=symbols, stimfile="mgold_65_6532_psk_60hz.png")
```

Now that the system is up and running, you can go through the following steps to use the BCI!

1. EEG headset Setup

   Prepare a headset such that it follows the MindAffect headset layout.pdf in our Headset repository or prepare the headset delivered with your kit by following MindAffect headset setup.pdf

2. Signal Quality

   Check the signal quality by pressing 0 in the main menu. Try to adjust the headset until all electrodes are green, or noise to signal ratio is below 5. You can try to improve the signal for an electrode by pressing it firmly into your head. After releasing pressure, wait a few seconds to see if the signal improves. If not, remove the electrode, and apply more water to the sponge. The sponges should feel wet on your scalp. If the noise to signal ratio does not improve by adjusting the headset, try to distance yourself from power outlets and other electronics.

3. Calibration

   Start calibration by pressing 1 in the main menu. Continue to follow the on-screen instructions.

4. Feedback

   You are now ready to try out the BCI by either selecting Copy-spelling (2) or Free-spelling (1)!

Struggling to get the system to work? Consult our FAQ section for info on how to improve calibration accuracy, prediction performance, and more!

SHUTDOWN

At this point, even though the presentation has completed, the background processes which run the hub, acquisation and decoder are still running. To ensure they are stopped cleanly it is **always** a good idea to shut them down correctly, as follows:

```
[ ]: # shutdown the background processes
# N.B. only needed if something went wrong..
mindaffectBCI.online_bci.shutdown(hub_process, acq_process, decoder_process)
```

Alternative: Run all components with 1 command

If you just want to quickly run the decoding part of the BCI without presentation/output, using a pre-defined configuration you can do that easily by. 1. loading a configuration file 2. running the BCI This is demonstrated here.

```
[ ]: config = mindaffectBCI.online_bci.load_config('noisetag_bci')
# uncomment this line to use fakedata
#config['acquisition']='fakedata'
mindaffectBCI.online_bci.run(**config)
```

Do you not want to run this whole notebook everytime when using the BCI?

Simply run it from your command prompt:

```
python3 -m mindaffectBCI.online_bci
```

Or with a JSON configuration file:

```
python3 -m mindaffectBCI.online_bci --config_file noisetag_bci.json
```

Going Further

1. Try other BCI types using our alternative configuration files.
   - noisetag.json : example for a noise-tagging (or c-VEP) BCI (Default)
   - rc5x5.json : example for a classic visual P300 odd-ball type BCI with row-column stimulus.
   - ssvep.json : example for a classic steady-state-visual-response BCI.
2. Write your own presentation system by following this guide (https://mindaffect-bci.readthedocs.io/en/latest/simple_presentation_tutorial.html)
3. Write your own output system to make *interesting* things happen when a brain control is activated following this guide (https://mindaffect-bci.readthedocs.io/en/latest/simple_output_tutorial.html)
4. Build your own BCI following this guide to develop your own compents. (https://mindaffect-bci.readthedocs.io/en/latest/first_run.html)

[ ]:

[ ]:

3.7.2 Creating a Simple Output Module

An output module listens for selections from the MindAffect decoder and acts on them to create some output. By the end of this tutorial you will be able to: * make a simple output module which prints "Hello World" when the presentation 'button' with ID=1 is selected. * know how to make your own output modules to perform arbitary actions under brain control

Before starting this tutorial please read the how an evoked bci works **tutorial to gain a general understanding of the components of this BCI. And the *quickstart tutorial* or *DIY Brain Computer Interface* for how they are used in the mindaffectBCI. **

The non-Jupyter Notebook version of this code and other output examples can be found in the examples/output *directory*

First, we import the module responsible for translating selections to output, and create the utopia2output object:

[1]:
```
# Import the utopia2output module
from mindaffectBCI.utopia2output import Utopia2Output

u2o=Utopia2Output()
```

Defining the Output function

Now we define a function to print hello-world.

[2]:
```
def helloworld(objID):
   print("hello world")
```

Connecting the Output function

And connect it so it is run when the object with ID=1 is selected. You can create as many output functions as needed, and connect them to different object ID's by simply adding them to the dictionary as *key-value* pairs.

[3]:
```
# set the objectID2Action dictionary to use our helloworld function if 1 is selected
u2o.objectID2Action={ 1:helloworld }
```

Start the BCI decoder in the background.

To successfully test your presentation module it is important to have the other components of the BCI running. As explained in the *quickstart tutorial*, additionally to the presentation we build here, we need the Hub, Decoder, and Acquisition components for a functioning BCI.
For a quick test (with fake data) of this presentation module you can run all these components with a given configuration file using.

N.B. if you run directly in this notebook, *don't forget to shutdown the decoder* at the end.

[ ]:
```
import mindaffectBCI.online_bci
config = mindaffectBCI.online_bci.load_config('fake_recogniser.json')
mindaffectBCI.online_bci.run(**config)
```

Alternatively you can run this config from the command line with:

```
python3 -m mindaffectBCI.online_bci --config_file fake_recogniser.json
```

Or from you Anaconda environment:

```
python -m mindaffectBCI.online_bci --config_file fake_recognizer.json
```

See our tutorial *Running Custom Presentation* **to set-up a BCI using your own Presentation module**

Connect and Run

We now connect our output module to a running decoder and run the main loop. Selection of ID=1 will result in printing "Hello World" to this prompt.

[ ]:
```
# connect
u2o.connect()

# run the main loop
u2o.run()
```

3.7.3 Creating a simple Presentation Module

This tutorial walks you through how to make your own **Presentation** module for the BCI. By the end of this tutorial you will be able to: * design your own user interface screen with virtual 'buttons' selectable by brain signals * connect to the decoder and run through the calibration to train the BCI * use your designed user-interface screen to select buttons the on-screen buttons

Before running this tutorial you should have read how an evoked bci works to get an overview of how this BCI works and it's main components, and run through *quickstart tutorial* to quickly test your installation and try the BCI.

- Note: The non-Jupyter Notebook version of this code and other presentation examples can be found in the* examples/presentation directory

The presentation module is responsible for displaying the user-interface to the user with flickering options they can select from with brain signals (see how an evoked bci works for more information). In order for the presentation flickering to generate the strongest possible brain response, and hence maximise the BCI performance, it must display the correct stimuli to the user with precise timing and communicate this timing information to the MindAffect decoder. Further, in order to train the decoding model presentation happens in two different modes:

- *calibration* mode where we cue the user where to attend to obtain correctly labelled brain data to train the machine learning algorithms in the decoder and
- *prediction* mode where the user actually uses the BCI to make selections.

The *noisetag* module provides a number of tools to hide this complexity (different modes, timestamp recording) from the application developers. Using the most extreme of these all the application developer has to do is provide a function to *draw* the display as instructed by the noisetag module. To use it, we import the module and create the noisetag object:

```
[ ]: from mindaffectBCI.noisetag import Noisetag, sumstats
     nt = Noisetag()
```

The Draw function

We will now write a function to draw the screen. Here we use the python gaming library pyglet to draw 2 squares on the screen, with the given colors.

```
[ ]: import pyglet
     # define a simple 2-squares drawing function
     def draw_squares(col1,col2):
         # draw square 1: @100,190 , width=100, height=100
         x=100; y=190; w=100; h=100;
         pyglet.graphics.draw(4,pyglet.gl.GL_QUADS,
                              ('v2f',(x,y,x+w,y,x+w,y+h,x,y+h)),
                   ('c3f',(col1)*4))
         # draw square 2: @440,100
         x=640-100-100
         pyglet.graphics.draw(4,pyglet.gl.GL_QUADS,
                              ('v2f',(x,y,x+w,y,x+w,y+h,x,y+h)),
                   ('c3f',(col2)*4))
```

Updating the Display

Now we write a function which, 1) asks the noisetag framework how the selectable squares should look, 2) updates the noisetag framework with information about how the display was updated.

```
[ ]: # dictionary mapping from stimulus-state to colors
state2color={0:(.2,.2,.2), # off=grey
             1:(1,1,1),    # on=white
             2:(0,1,0),    # cue=green
             3:(0,0,1)}    # feedback=blue
def draw(dt):
    '''draw the display with colors from noisetag'''
    # send info on the *previous* stimulus state, with the recorded vsync time (if␣
↪available)
    fliptime = window.lastfliptime if window.lastfliptime else nt.getTimeStamp()
    nt.sendStimulusState(timestamp=fliptime)
    # update and get the new stimulus state to display
    try :
        nt.updateStimulusState()
        stimulus_state,target_state,objIDs,sendEvents=nt.getStimulusState()
    except StopIteration :
        pyglet.app.exit() # terminate app when noisetag is done
        return

    # draw the display with the instructed colors
    if stimulus_state :
        draw_squares(state2color[stimulus_state[0]],
                     state2color[stimulus_state[1]])

    # some textual logging of what's happening
    if target_state is not None and target_state>=0:
        print("*" if target_state>0 else '.',end='',flush=True)
    else:
        print('.',end='',flush=True)
```

Integrating Output

As a final step we integrate the behaviour of the output module created in the *Simple Output Tutorial* by attaching a selection callback which will be called whenever a selection is made by the BCI. For now we will use the "Hello World" example created in the Output Tutorial and add a function that prints the object ID of our second square when selected.

```
[ ]: def helloworld(objID):
    print("hello world")

def printID(objID):
    print("Selected: %d"%(objID))

# define a trivial selection handler
def selectionHandler(objID):
    selection_mapping = {
        1:helloworld,
        2:printID
    }
    func = selection_mapping.get(objID)
    func(objID)
```

Timing Accuracy

Now, we need a bit of python hacking. Because our BCI depends on accurate timelock of the brain data (EEG) with the visual display, we need to have accurate time-stamps for when the display changes. Fortunately, pyglet allows us to get this accuracy as it provides a flip method on windows which blocks until the display is actually updated. Thus we can use this to generate accurate time-stamps. We do this by adding a time-stamp recording function to the windows normal flip method with the following magic:

[ ]:
```
import types

def timedflip(self):
    '''pseudo method type which records the timestamp for window flips'''
    type(self).flip(self) # call the 'real' flip method...
    self.lastfliptime=nt.getTimeStamp()
```

Next, we initialize the window to display the stimulus, and setup the flip-time recording for it. Be sure that you have vsync turned-on. Many graphics cards turn this off by default, as it (in theory) gives higher frame rates for gaming. However, for our system, frame-rate is less important than exact timing, hence always turn vsync on for visual Brain-Compuber-Interfaces!

Note: always set `fullscreen=True` when using the presentation module to improve screen timing accuracy. We set it to `False` here so the tutorial stays visible when the stimulus is running.

Note: When running in a notebook the pyglet window always starts minimized – so if you can't see it check your task bar.

[ ]:
```
# Initialize the drawing window
# make a default window, with fixed size for simplicty, and vsync for timing
config = pyglet.gl.Config(double_buffer=True)
window = pyglet.window.Window(fullscreen=False, width=640,height=480, vsync=True,␣
↪config=config)

# Setup the flip-time recording for this window
window.flip = types.MethodType(timedflip,window)
window.lastfliptime=None
```

Start the BCI decoder in the background.

To successfully test your presentation module it is important to have the other components of the BCI running. As explained in the *quickstart tutorial*, additionally to the presentation we build here, we need the Hub, Decoder, and Acquisition components for a functioning BCI.

For a quick test (with fake data) of this presentation module you can run all these components with a given configuration file using.

N.B. if you run directly in this notebook, *don't forget to shutdown the decoder* at the end.

[ ]:
```
import mindaffectBCI.online_bci
config = mindaffectBCI.online_bci.load_config('fake_recogniser.json')
mindaffectBCI.online_bci.run(**config)
```

Alternatively you can run this config from the command line with:

```
python3 -m mindaffectBCI.online_bci --config_file fake_recogniser.json
```

Or from you Anaconda environment:

```
python -m mindaffectBCI.online_bci --config_file fake_recognizer.json
```

See our tutorial *Running Custom Presentation* to set-up a BCI using your own Presentation module

Run the Experiment!

To run the experiment we connect to the Hub, add our selection handler, tell the noisetag module to run a complete BCI 'experiment' with calibration and feedback mode, and start the pyglet main loop.

[ ]:
```
# Initialize the noise-tagging connection
nt.connect(timeout_ms=5000)
nt.addSelectionHandler(selectionHandler)

# tell the noisetag framework to run a full : calibrate->prediction sequence
nt.setnumActiveObjIDs(2)
nt.startExpt(nCal=4,nPred=10,duration=4)

# run the pyglet main loop
pyglet.clock.schedule(draw)
pyglet.app.run()
```

Shutdown the decoder

[ ]:
```
import mindaffectBCI.online_bci
mindaffectBCI.online_bci.shutdown()
```

3.7.4 Running Custom Presentation

This tutorial shows how to integerate a custom presentation module (such as the custom presentation module made in the *Simple Presentation Tutorial*), with the rest of the BCI run with the `online_bci.py` module. By the end of this tutorial you will know: * How to start the mindaffectBCI **without** automatically starting a presentation module * How to run your presentation module * How to cleanly shutdown the mindaffetBCI when you finish

Before running this tutorial you should have read how an evoked bci works and run through the *Simple Presentation Tutorial* to understand how to build a custom presentation module.

[ ]:
```
# Import the mindaffectBCI decoder and other required modules.
%load_ext autoreload
%autoreload 2
%gui qt
import mindaffectBCI.online_bci
```

Start the EEG processing components

That is start the; 1) Hub, 2) Acquisation, 3) Decoder - by starting the `online_bci` script with presentation disabled.

For more information on these components and why we need them please consult the *Quickstart Tutorial*

[ ]:
```
# load the config-file to use.  Here we use the default noisetag config file:
config = mindaffectBCI.online_bci.load_config("noisetag_bci")
```

Set the Presentation system to `None` and then start the online_bci.

```
[ ]: config['presentation']=None
     #uncomment the following line to use to the fakedata acquisition
     config['acquisition']='fakedata'
     mindaffectBCI.online_bci.run(**config)
```

Check all is running correctly.

```
[ ]: mindaffectBCI.online_bci.check_is_running()
```

Run the Custom Presentation

We run our custom presentation module by first importing and then running it.

```
[ ]: # import and run the presentation
     from mindaffectBCI.examples.presentation import minimal_presentation
```

Shutdown the decoder

```
[ ]: mindaffectBCI.online_bci.shutdown()
```

3.7.5 How to build your own optical sensor

Equipment

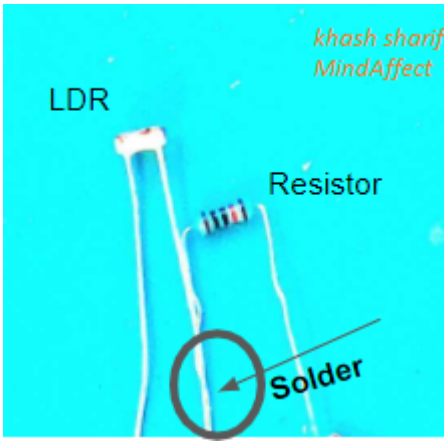
- **basic soldering kit**
  - Soldering iron
  - Soldering tin
  - Soldering mat
  - Wires with 3 different colors
  - Shrink tubes
- 100k ohm resistor
- **photoresistor(LDR)**
  - GL5528 LDR or GL5537 LDR
- **3v DC power supply of your choice (any of the following options would work fine)**
  - 2x AAA batteries in parallel
  - 3v GPIO pins of the OpenBCI board
  - 3v LiPo battery

Circuit design concept

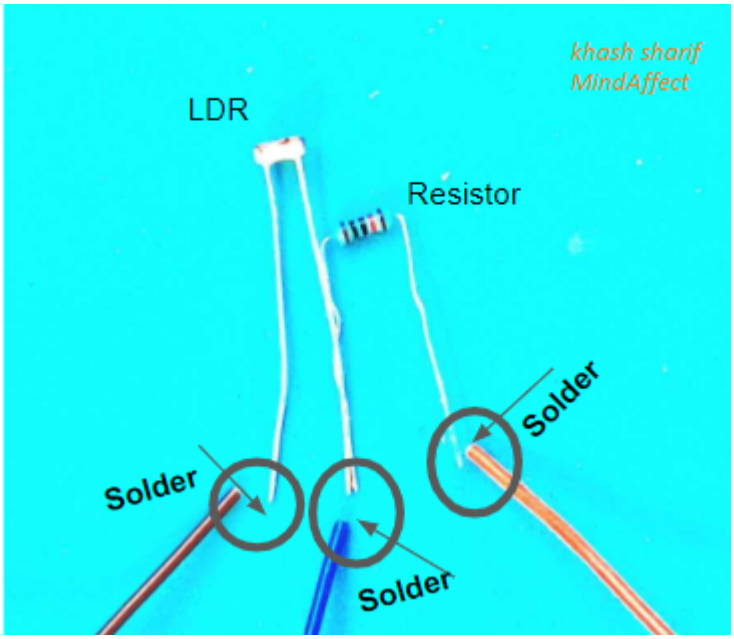
The circuit is a simple voltage divider, consisting of a number resistor and a LDR. The LDR's resistance is a function of light exposure. The resistance is reciprocally proportional to light intensity. Thus, by exposing the LDR to any of the visual noise tags (e.g. the flickering buttons on the screen of your computer) The output voltage mimics the behavior of the flickering pattern.

Steps to build the sensor

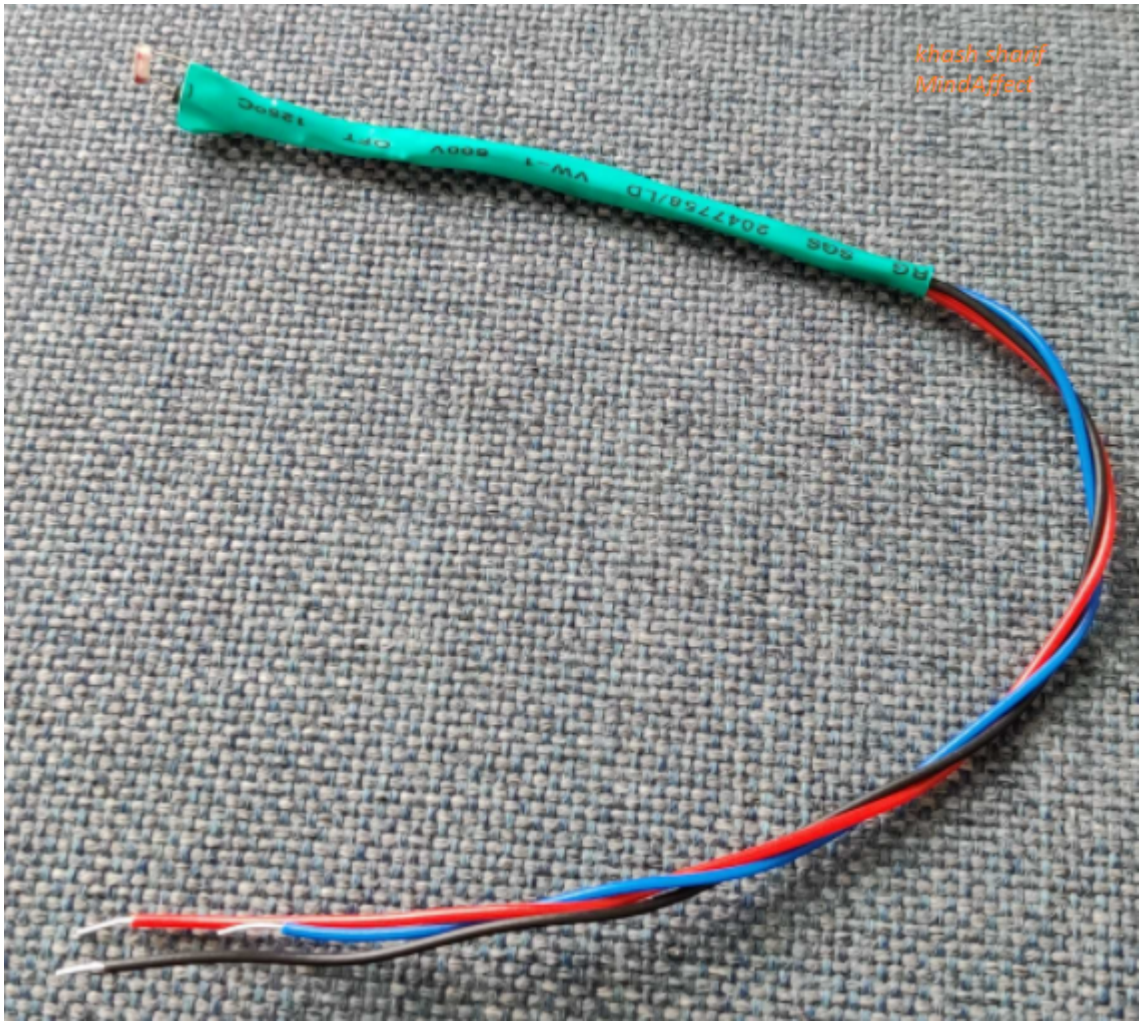
1. Solder the resistor and the LDR together.



2. Solder the red, blue and black wires as demonstrated below.



3. Add shrink tubes at the soldering locations and apply heat to insulate the connections.
4. You have yourself an optical sensor!



How to use this sensor with an OpenBCI board

First, the sensor must be connected to the OpenBCI board as described below.

- Red wire –> vDD OpenBCI board.
- **Blue wire –> signal pin of openbci. You can use any free channel as your signal pin.**
  - Channels 1-8 for the openBCI cyton board.
  - Channels 1-4 for the openBCI ganglion board.

set the board to bipolar mode

- **For the cyton board, go to *pymindaffectBCI/mindaffectBCI* directory and open the online_bci.json file. Next, enable trigge**

```
{
    "acquisation":"brainflow",

    "acq_args":{
        "board_id":0,
        "triggerCheck": 1
        "serial_port":"com3"
    },

    "decoder":"mindaffectBCI.decoder.decoder",
    "decoder_args":{
        "stopband" : [[45,65],[0,3],[25,-1]],
        "out_fs" : 100,
        "evtlabs" : ["re","fe"],
        "tau_ms" : 450,
        "calplots" : true,
        "predplots" : false
    },
```

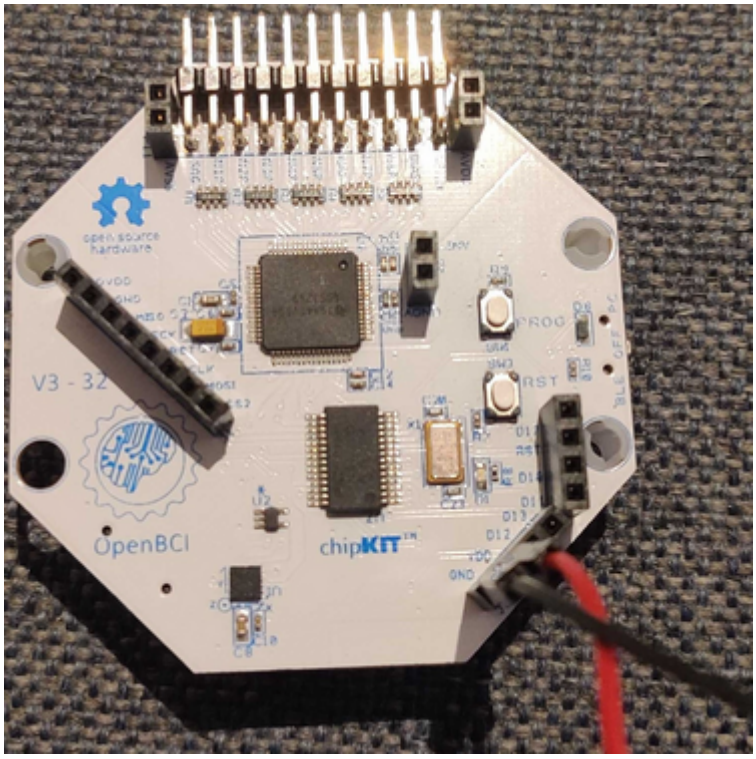
khash sharif
MindAffect

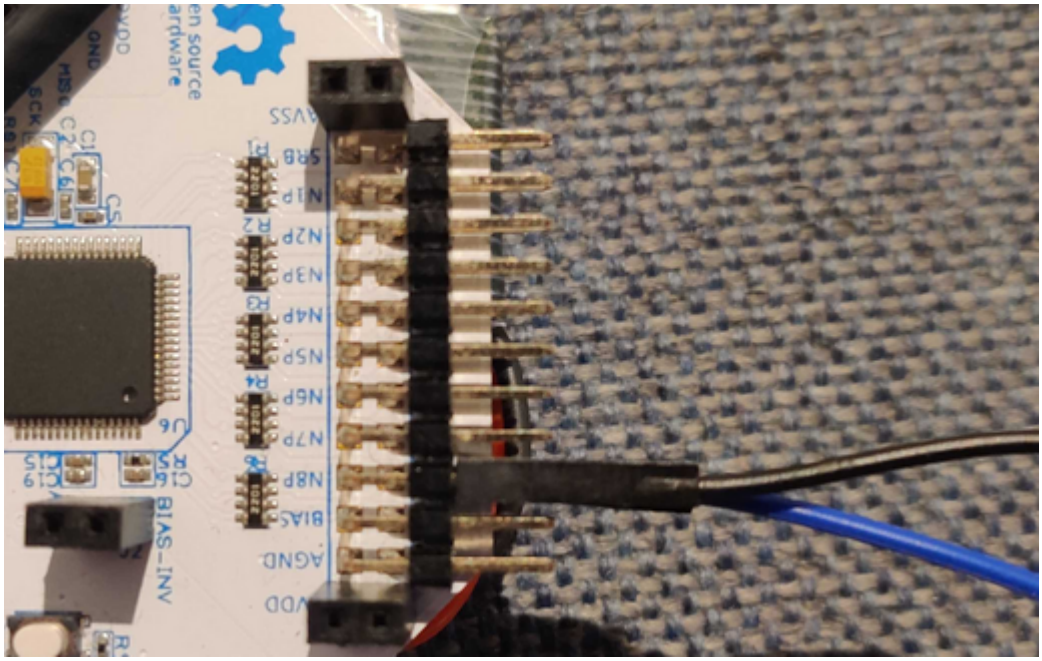
- For the ganglion board, follow this guide and set the switches to **DOWN** position.

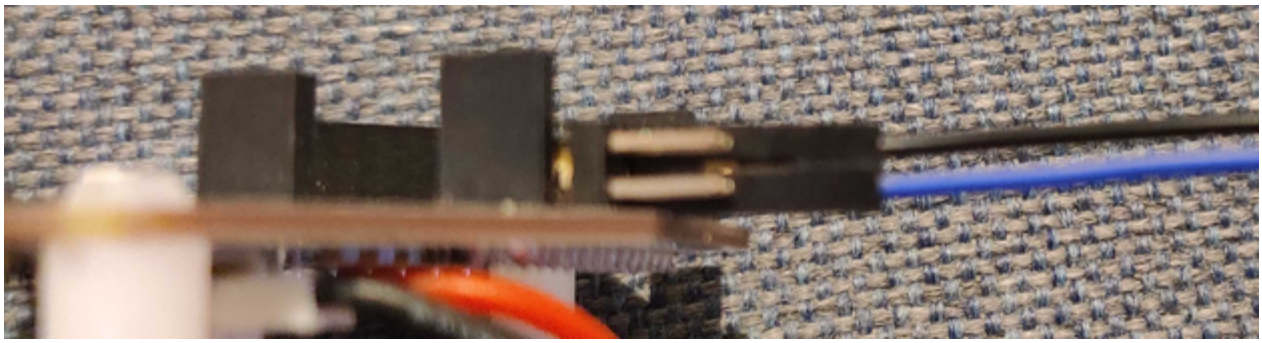
Then the blue wire should be connected to

- The lower pin of cyton channel of choice (by default this is set to channel 8)
- The upper pin of ganglion channel of choice

The black wire should be connected to the *GND* gpio pin and the ground signal pin (in bipolar mode, the lower pin of ganglion and the upper pin of cyton)


V3 - 32
OpenBCI
chipKIT





Testing the functionality

To quickly test the optical sesnor

1. Connect the OpenBCI to your PC and power up the board
2. Connect the Optical sensor to the OpenBCI board (as discussed above)
3. run the MindaffectBCI program as usual
4. start the MindaffectBCI presentation
5. perform calibration by placing the optical sesnor in front of the cued buttons
6. go to prediction mode. The system should detect any button that is put in front of the optical sensor (as demonstarted below)

3.7.6 Checking stimulus time-lock quality

One very important consideration when using an evoked-response Brain Computer Interface (ERP-BCI) is ensuring an accurate *time-lock* between the stimuli and the measured EEG. As a minimum this time-lock needs to be accurate enough to unambiguously identify which stimulus a brain response comes from. Thus, for example if we are using a 60Hz display to present stimuli, this means a new stimulus can happen very 1/60s = 16ms, so to ensure we can identify which brain response corresponds to which stimulus we require a time-lock accuracy between the display and the measured EEG of <16ms.

This tutorial shows you how to run an hardware test of your BCI design to objectively measure the time-lock precision quality. By the end of this tutorial you will be able to: * use an opto-resistor to directly link your stimulus presentation system to your EEG measurment system * run a test BCI run to get data to check the time-lock quality * generate visualizations of the time-lock quality data which show: the general trigger check quality, the single-trial stimulus->trigger connection in time, and the trigger response model used by the BCI * interpert the results of this trigger check quality plot to check if the quality is sufficient

Introduction

So how do you ensure you have a sufficient time-lock quality? One approach is to 'just-try-it' with a brain. If you are lucky this will 'just-work' – in which case you can conclude that your timing was probably sufficiently good. However, what if it doesn't work? Then you do cannot tell where the problem was, was it the cap-fitting, a stimulus bug, a decoder bug, or a poor time-lock.

To avoid these types of situations it is better to debug, and measure the time-lock in isolation.

We are specifically concerned with the time-lock quality between: 1. the *stimulus* generated by our presentation software on some hardware stimulus device (most likely a screen) and 2. the EEG measured from some hardware acquisition device by our decoder

Thus, to do an effective time-lock we need to: * Run the stimulus presentation system * Run the decoder and acquisition device * Add a hardware *very low latency* connection between the stimulus and acquisition system to *inject* stimulus signals into the acquisition device * Run some additional analysis to measure and visualize the timing accuracy

This tutorial walks through how to do such a measurement for a visual-presentation system using an *opto-resistor* to couple the stimulus to the EEG system.

To do this test you will need: 1. A visual stimulus presentation system. 2. A supported EEG measurement system. Here we will use the openBCI cyton, but the same technique should work for any EEG device. 3. An opto-resistor connecting the screen to the amplifier. 4. This notebook to run the trigger-check script, which generates trigger checking results

The main steps in performing the test are: 1. Connect the opto-resistor to the display 2. Run your BCI test. 3. Analyse the gathered results. (Though this can also be done on-line while the BCI is running.)

Step 1: Connect the opto-resistor to the display

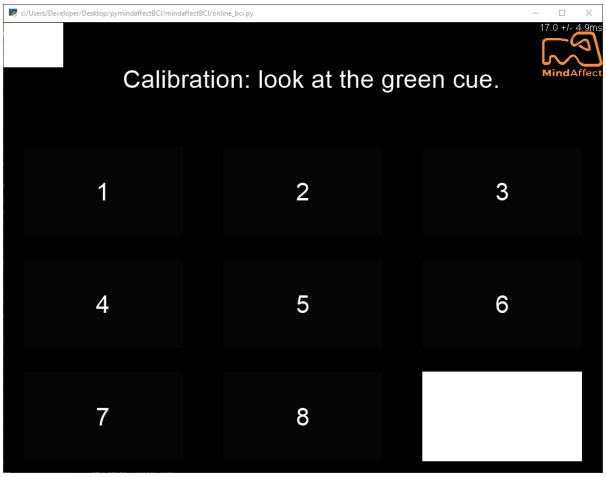
You can find the instructions for building an opto-resistor and connecting it to the EEG amplifier here.

Note: to use the opto-sensor for time-lock quality, the opto-sensor *must* measure the *target* stimulus.

That is the button which is cued green during calibration or cued feedback. To make this easier in our default

selectionMatrix.py

there is a special 'opto-sensor' location at the top-left of the screen (see below) which *always* shows a copy of the target stimulus – so you can simply fix the opto-resistor at that location.



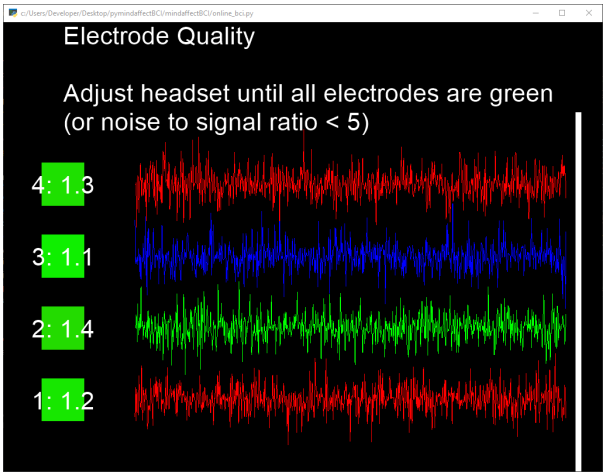
Step 2: Run your BCI test run

After connecting the display to the amplifier. The next stage is to run the BCI with the opto-resistor connected to get the timing data. As we want to test the whole on-line processing loop we do this by simply running the full BCI stack, where basically the opto-resistor replaces the brain with a 'perfect' response brain.

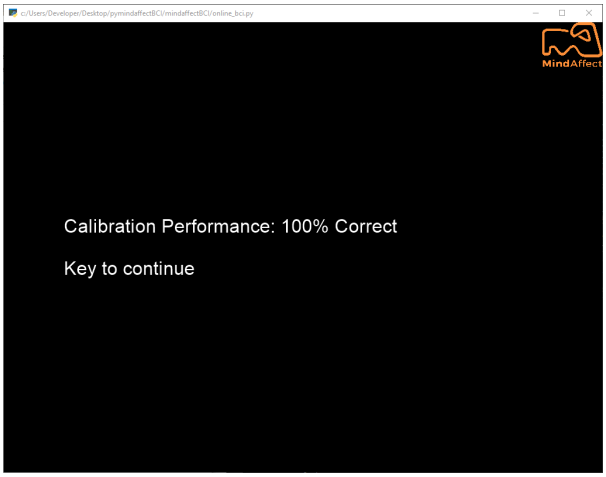
To run the BCI we will use the `online_bci.py` script with the `trigger_check.json` configuration file.

Note: if you do not have the EEG or opto-resistor hardware, you can simulate this run by using the `acquisition='fakedata'` option, which will use the fake-data EEG simulator, which also include a software trigger injector.

After the BCI has started: * Press 0 to go to the signal viewer. Here you can quickly check if your opto-resistor is working correctly by changing the display brightness and looking for the change in the appropriate EEG channel.



- Press a key to exit the signal viewer.
- Press 1 to enter cued calibration. The system will now do 10 calibration trials of 4.2s each, and then fit the BCI model. If your opto-resistor (and the BCI) is working correctly, it should tell you the *calibration performance was 100%*.



- Press a key to return to the main menu. You can now press 2 to enter cued prediction if you wish, and test the trained BCI model. Alternatively, as you have now got sufficient data for the time-lock analysis you can press q or escape to quit.

[ ]:
```
import mindaffectBCI.online_bci
# load the configuration file
config = mindaffectBCI.online_bci.load_config('trigger_check')

# for debugging with the fakedata source uncomment this line!
```

(continues on next page)

(continued from previous page)

```
#config['acquisition']='fakedata'

# run with the loaded configuration
mindaffectBCI.online_bci.run(**config)
```

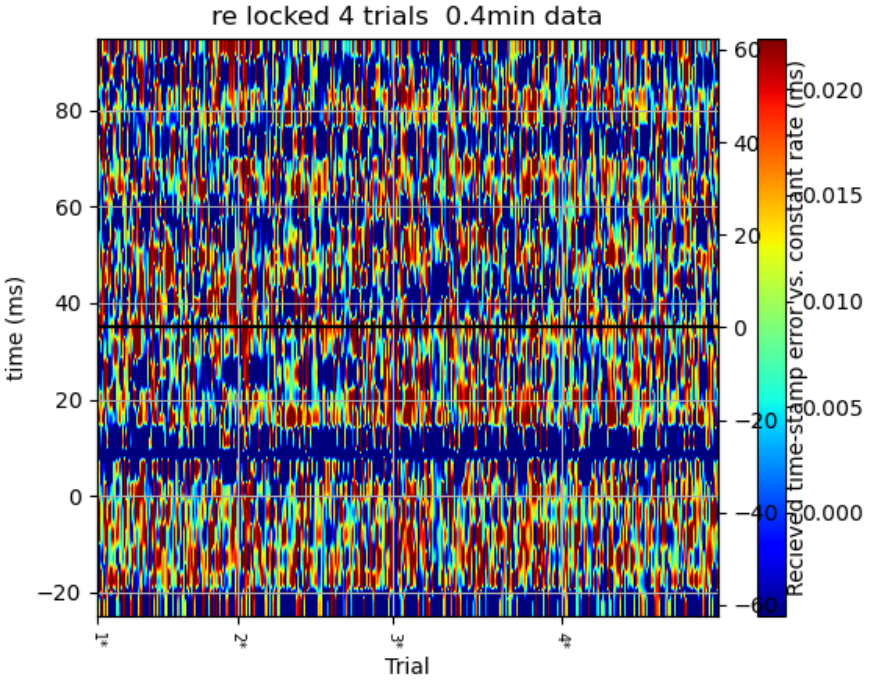
Step 3: Analyse the time-lock analysis data

If the above test ran correctly the system will have saved a data file in the `logs` directory of the project with a name like `mindaffectBCI_YYMMDD_HHMM.txt` containing all the information about your experiment. This can be used for offline-analysis to explore and refine the model fitting process as explained in the offline analysis tutorial.

Here we will just do a trigger-check analysis. For this we need to know which save file to load. Fortunately, as a convenience for this situation if we use the filename – the system will automatically load the most recent save file from the `logs` directory – which is the one we just generated.

When this check runs it will generate 3 plots visualizing the trigger check data:

1. *Trigger Check*: This plot shows as an image the measured EEG response for every *stimulus onset* (i.e. when the stimuli gets brighter) for every trial. This is displayed as an image where the vertical axis is time since the stimulus, and the horizontal axis are the onsets within each trial, with color showing the magnitude of the EEG. This is the key plot for determining the timing accuracy.

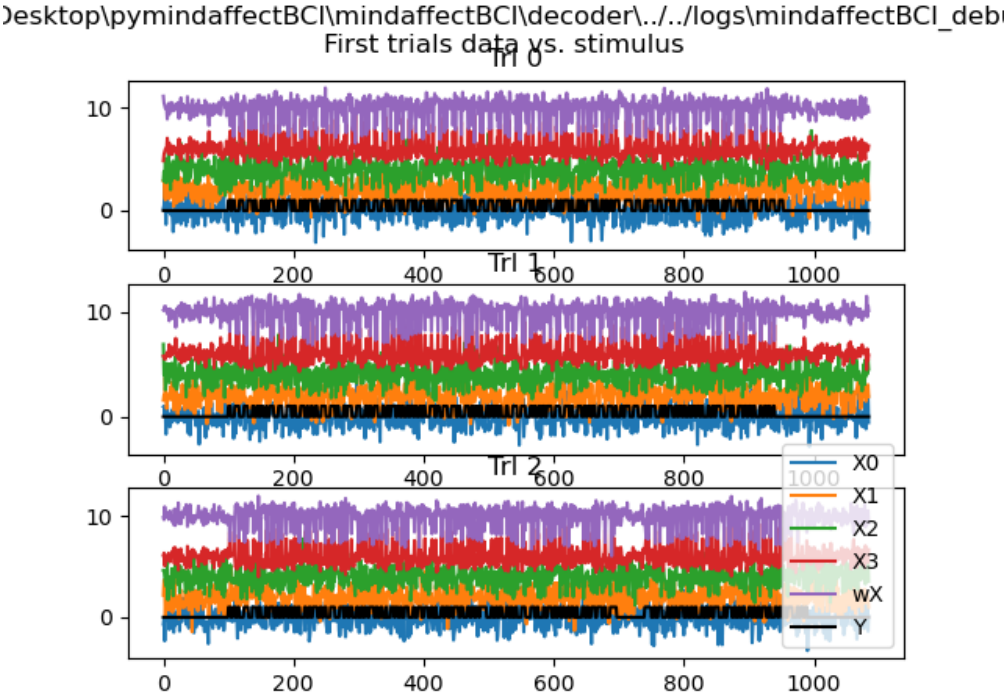


If your timing is accurate you should see a clear *horizontal* band at some time after 0ms – here it's a blue line at about 10ms (in fact due to filtering artifacts you may see more than one such band). By zooming in the y-axis you should see that:

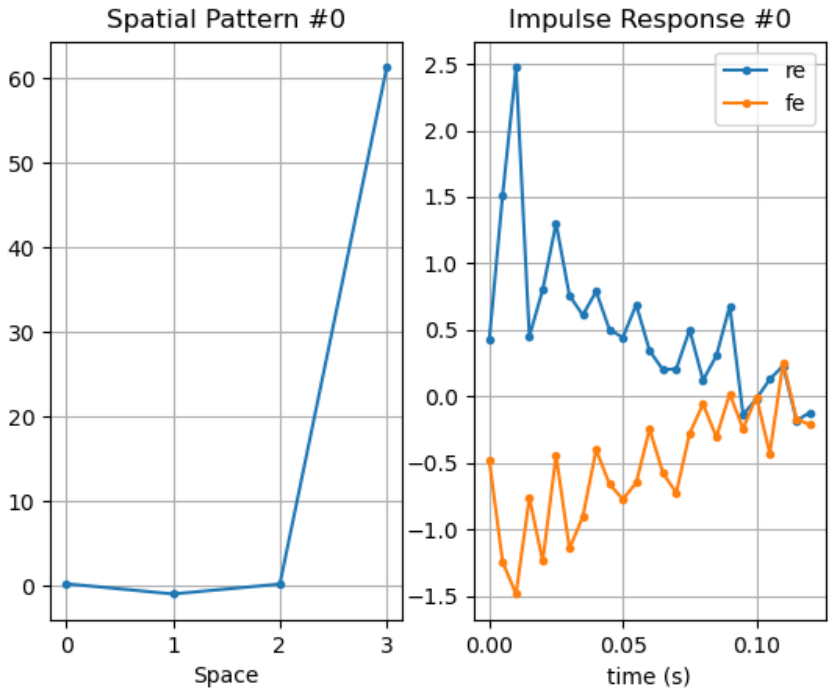
- The bands are *perfectly* horizontal, with no jitter. If there is timing jitter you will see the start of the band move up and down between trials. Here you can see there is a jitter of about +/-5ms – which is 1 sample as the simulated amplifier runs at 200Hz.

- The bands have no slope or steps. Slopes imply that the stimulus timing is *drifting* with respect to the EEG measurements. Steps imply the stimulus timing is suddenly increasing and decreasing in latency with respect to its nominal time-stamp. In particular we have noticed that steps are common for displays as the operating system (particularly windows) adds and removes frames from the graphics system to draw animations and blurs.

2. *First Trials*: This plot shows the raw time series for the first 3 trials of the saved data. In addition it shows the stimulus data in `Y` (*black*) and predicted stimulus data generated by the fitted model in `wX`. If your opto-resistor is working correctly you should clearly see the opto signal in the appropriate channel of X (channel 8 if using the cyton.)



3. *Classifier Model*. This plot shows the parameters of the model trained by the classifier to identify the trigger response. In particular the spatial pattern should match that of the opto-resistor channel if the model is correct.



[ ]:
```
from mindaffectBCI.decoder.trigger_check import trigger_check
trigger_check('-')
```

3.7.7 Offline analysis of a mindaffectBCI savefile

So you have successfully run a BCI experiment and want to have a closer look at the data, and try different analysis settings?

Or you have a BCI experiment file from the internet, e.g. MOABB, and want to try it with the mindaffectBCI analysis decoder?

Then you want to do an off-line analysis of this data!

This notebook shows how to such a quick post-hoc analysis of a previously saved dataset. By the end of this tutorial you will be able to: * Load a mindaffectBCI savefile * generate summary plots which show; the per-channel grand average spectrum, the data-summary statistics, per-trial decoding results, the raw stimulus-resonse ERPs, the model as trained by the decoder, the per-trial BCI performance plots. * understand how to use these plots to identify problems in the data (such as artifacts, excessive line-noise) or the BCI operation * understand how to change analysis parameters and the used classifier to develop improved decoders

[ ]:
```
import numpy as np
from mindaffectBCI.decoder.analyse_datasets import debug_test_dataset
from mindaffectBCI.decoder.offline.load_mindaffectBCI  import load_mindaffectBCI
import matplotlib.pyplot as plt
%matplotlib inline
%load_ext autoreload
%autoreload 2
plt.rcParams['figure.figsize'] = [12, 8] # bigger default figures
```

Specify the save file you wish to analyse.

You can either specify: * the full file name to load, e.g. '~/Downloads/mindaffectBCI_200901_1154.txt' * a wildcard filename, e.g. '~/Downloads/mindaffectBCI*.txt', in which case the **most recent** matching file will be loaded. * `None`, or '-', in which case the most recent file from the default `logs` directory will be loaded.

```
[ ]: # select the file to load
     #savefile = '~/../../logs/mindaffectBCI_200901_1154_ssvep.txt'
     savefile = None # use the most recent file in the logs directory
     savefile = 'mindaffectBCI_exampledata.txt'
```

Load the *RAW*data

Load, with minimal pre-processing to see what the raw signals look like. Note: we turn off the default filtering and downsampling with `stopband=None, fs_out=None` to get a true raw dataset.

It will then plot the grand-aver-spectrum of this raw data. This plot shows for each EEG channel the signal power across different signal frequencies. This is useful to check for artifacts (seen as peaks in the spectrum at specific frequencies, such as 50hz), or bad-channels (seen as channels with excessively high or low power in general.)

During loading the system will print some summary information about the loaded data and preprocessing applied. Including: * The filter and downsampling applied * The number of trails in the data and their durations * The trail data-slice used, measured relative to the trial start event * The EEG and STIMULUS meta-information, in terms of the array shape, e.g. (13,575,4) and the axis labels, e.g. (trials, time, channels) respectively.

```
[ ]: X, Y, coords = load_mindaffectBCI(savefile, stopband=None, fs_out=None)
     # output is: X=eeg, Y=stimulus, coords=meta-info about dimensions of X and Y
     print("EEG: X({}){} @{}Hz".format([c['name'] for c in coords],X.shape,coords[1]['fs
     ↪']))
     print("STIMULUS: Y({}){}".format([c['name'] for c in coords[:-1]]+['output'],Y.shape))
     # Plot the grand average spectrum to get idea of the signal quality
     from mindaffectBCI.decoder.preprocess import plot_grand_average_spectrum
     plot_grand_average_spectrum(X, fs=coords[1]['fs'], ch_names=coords[-1]['coords'],␣
     ↪log=True)
```

Reload the data, with standard preprocessing.

This time, we want to analysis the loaded data for the BCI signal. Whilst we can do this after loading, to keep the analysis as similar as possible to the on-line system where the decoder only sees pre-processed data, we will reload wand apply the pre-processing directly. This also has the benefit of making the datafile smaller.

To reproduce the pre-processing done in the on-line BCI we will set the pre-processing to: * temporally filter the data to the BCI relevant range. Temporal filtering is a standard technique to remove sigal frequencies which we know only contain noise. For the noise-tag brain response we know it is mainly in the frequency range from 3 to about 25 Hz. Thus, we specifcy a bandpass filter to only retain these frequencies with: `stopband=(3,25,'bandpass')` * The orginal EEG is sampled 250 times per second. However, the BCI relevant signal changes at most at 25 times per second, thus the EEG is sampled much more rapidly than needed – so processing it takes undeeded computational resources. Thus, we downsmaple the data to save some computation. To avoid signal-artifacts, as a general 'rule of thumb' you should downsample to about 3 times your maximum signal frequency. In this case we use an output sample rate of 4 times, or 100 h with: `fs_out=100`

```
[ ]: X, Y, coords = load_mindaffectBCI(savefile, stopband=(3,25,'bandpass'), fs_out=100)
     # output is: X=eeg, Y=stimulus, coords=meta-info about dimensions of X and Y
```

(continues on next page)

(continued from previous page)

```
print("EEG: X({}){} @{}Hz".format([c['name'] for c in coords],X.shape,coords[1]['fs
↪']))
print("STIMULUS: Y({}){}".format([c['name'] for c in coords[:-1]]+['output'],Y.shape))
```

Analyse the data

The following code runs the standard initial analysis and data-set visualization, in one go with some standard analysis parameters:

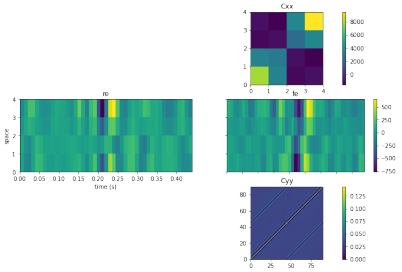
- tau_ms : the length of the modelled stimulus response (in milliseconds)
- evtlabs : the type of brain feaatures to transform the stimulus information into prior to fitting the model in this case
  - 're' -> rising edge
  - 'fe' -> falling edge

see `stim2event.py` for more information on possible transformations * rank : the rank of the CCA model to fit

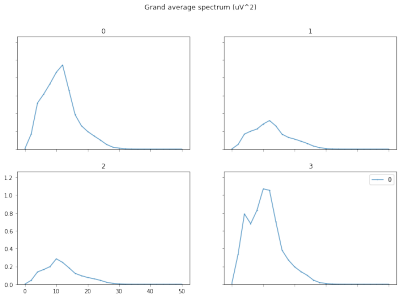
- model : the type of model to fit. 'cca' corrospends to the Cannonical Correlation Analysis model.

This generates many visualizations. The most important are:

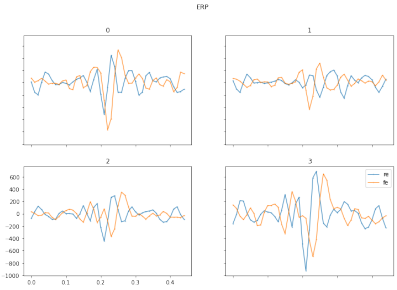
1. **Summary Statistics**: Summary statistics for the data with, This has vertically 3 sub-parts. row 1: Cxx : this is the spatial cross-correlation of the EEG channels row 2: Cxy : this is the cross-correlation of the stimulus with the EEG. Which for discrete stimuli as used in this BCI is essentially another view of the ERP. row 3: Cyy : the auto-cross covariance of the stimulus features with the other (time-delayed) stimulus features



2. **Grand Average Spectrum** : This shows for each data channel the power over different signal frequencies. This is useful to identify artifacts in the data, which tend to show up as peaks in the spectrum at different frequencies, e.g. high power below 3Hz indicate movement artifacts, high power at 50/60hz indicates excessive line-noise interference.



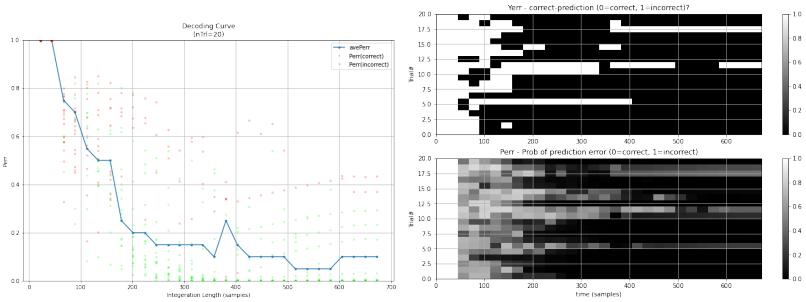
3. **ERP** : This plot shows for each EEG channel the averaged measured response over time after the triggering stimulus. This is the conventional plot that you find in many neuroscientific publications.



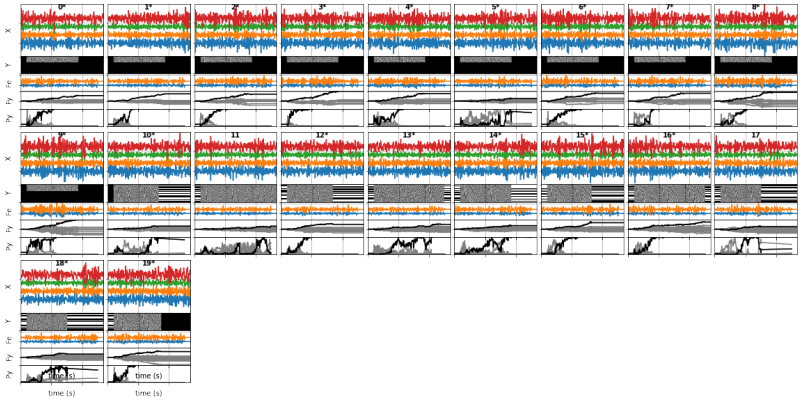
4. **Decoding Curve + Yerr** : The decoder accumulates information during a trial to make it's predictions better. These pair of plots show this information as a 'decoding curve' which shows two important things:

a) **Yerr** : which is the **true** error-rate of the systems predictions, with increasing trial time.

b) **Perr** : which is the systems own **estimation** of it's prediction error. This estimation is used by the system to identify when it is confident enough to make a selection and stop a trial early. Thus, this should ideally be as accurate as possible, so it's near 1 when Yerr is 1 and near 0 when Yerr is 0. In the DecodingCurve plot Perr is shown by colored dots, with red being Yerr=1 and green being Yerr=0. Thus, if the error estimates are good you should see red dots at the top left (wrong with short trials) and green dots at the bottom right (right with more data).



5. **Trial Summary** : This plot gives a direct trial-by-trial view of the input data and the BCI performance. With each trial plotted individually running from left to right top to bottom.



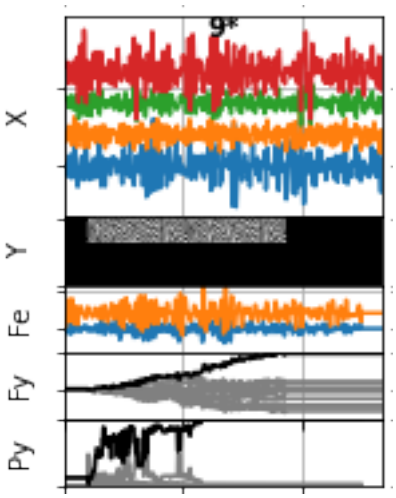
Zooming in on a single trial, we see that vertically it has 5 sub-parts:

```
a) **X** : this is the pre-processed input EEG data, with time horizontially, and␣
↪channels with different colored lines vertically.

b) **Y** : this is the raw stimulus information, with time horizontially and outputs␣
↪vertically.
```

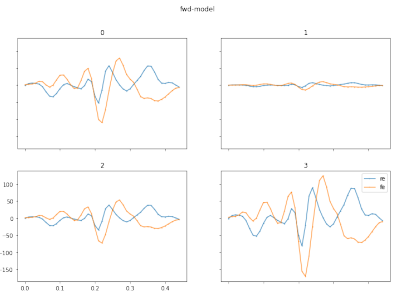
(continues on next page)

(continued from previous page)

```
c) **Fe** : this is the systems predicted score for each type of stimulus-event,␣
↪generated by applying the model to the raw EEG (e.g. 're','fe')

d) **Fy** : this is the systems _accumulated_ predicted score for each output,␣
↪generated by combining the predicted stimulus scores with the stimulus information.␣
↪ Here the **true** output is in black with the other outputs in grey.  Thus, if the␣
↪system is working correctly, the true output has the highest score and will be the␣
↪highest line.

e) **Py** : this is the systems **estimated** target probability for each output,␣
↪generated by softmaxing the Fy scores.  Again, the true target is in black with the␣
↪others in grey.  So if the system is working well the black line is near 0 when it
↪'s incorrect, and then jumps to 1 when it is correct.
```



6. *Model*: plot of the fitted model, in two sub-plots with: a) the fitted models spatial-filter – which shows the importance of each EEG channel, b) the models impulse response – which shows how the brain responds over time to the different types of stimulus event



[ ]:
```
clsfr=debug_test_dataset(X, Y, coords,
                         model='cca', evtlabs=('re','fe'), rank=1, tau_ms=450)
```

Alternative Analyse

The basic analysis system has many parameters you can tweak to test different analysis methods. The following code runs the standard initial analysis and data-set visualization, in one go with some standard analysis parameters:

tau_ms : the length of the modelled stimulus response (in milliseconds) evtlabs : the type of brain feaatures to transform the stimulus information into prior to fitting the model in this case ‘re’ -> rising edge ‘fe’ -> falling edge see

stim2event.py for more information on possible transformations rank : the rank of the CCA model to fit model : the type of model to fit. 'cca' corrospends to the Cannonical Correlation Analysis model. other options include: * 'ridge' = ridge-regression, * 'fwd' = Forward Modelling, * 'bwd' = Backward Modelling, * 'lr' = logistic-regression, * 'svc' = support vector machine

See the help for `mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence` or `mindaffectBCI.decoder.analyse_datasets.analyse_dataset` for more details on the other options.

Here we use a Logistic Regression classifier to classify single stimulus-responses into rising-edge (re) or falling-edge (fe) responses.

Note: we also include some additional pre-processing in this case, which consists of: * **whiten** : this will do a spatial whitening, so that the data input to the classifier is **spatially** decorrelated. This happens automatically with the CCA classifier, and has been found useful to suppress artifacts in the data. * **whiten_spectrum** : this will approximately decorrelate different frequencies in the data. In effect this flattens the peaks and troughs in the data frequency spectrum. This pre-processing also been found useful to suppress artifacts in the data.

Further, as this is now a classification problem, we set `ignore_unlabelled=True`. This means that samples which are not either rising edges or falling edges will not be given to the classifier – so in the end we train a simple binary classifier.

[ ]:
```
# test different inner classifier.  Here we use a Logistic Regression classifier to␣
↪classify single stimulus-responses into rising-edge (re) or falling-edge (fe)␣
↪responses.
debug_test_dataset(X, Y, coords,
                   preprocess_args=dict(badChannelThresh=3, badTrialThresh=None,␣
↪whiten=.01, whiten_spectrum=.1),
                   model='lr', evtlabs=('re', 'fe'), tau_ms=450, ignore_
↪unlabelled=True)
```

[ ]:
```

```

3.7.8 For DataScientists: How to analyse multiple datasets

So you have gathered a load of data using the mindaffectBCI from yourself on different days, or from a group of people. Now you want to see if you can tweak the analysis settings to get much better performance in later days. As you know some machine learning, you know that tweaking the settings on a single dataset will just lead to overfitting. So you want to try new settings on all the data you have so far to see if your changes generalize to new data.

In this tutorial you will learn;

- How to load all the datasets in a directory
- And apply the standard mindaffectBCI classification pipeline to them, to generate a summary of the performance on all the datasets.
- How to tweak different settings to see if you can improve the general performance.

Note: in addition to using this notebook to analysis data gathered locally, you can perform analysis directly on-line on our Kaggle dataset or download this data for local off-line analysis.

Imports and configuration setup.

```
[ ]: import numpy as np
     from mindaffectBCI.decoder.datasets import get_dataset
     import matplotlib.pyplot as plt
     from mindaffectBCI.decoder.analyse_datasets import analyse_dataset, analyse_datasets,␣
      ↪debug_test_dataset, debug_test_single_dataset
     %matplotlib inline
     %load_ext autoreload
     %autoreload 2
```

Set the directory you want to load the datasets for analysis from

```
[ ]: datasetsdir = "~/Desktop/datasets"
```

Check what datasets can be found in this directory

```
[ ]: dataset_loader, dataset_files, dataroot = get_dataset('mindaffectBCI',
      ↪exptdir=datasetsdir)
     print("Got {} datasets\n{}".format(len(dataset_files),dataset_files))
```

Set the parameters for the analysis to run. Specificially:

- `test_idx`: this is the trials to be used for performance evaluation in each dataset. All other trials are used for parameter estimation (in a nested cross-validation). Here we use `slice(10,None)` to say all trials after the 1st 10 are for testing.
- `loader_args` : dictionary of additional arguments to pass to the `load_mindaffectBCI.py` dataset loader, specifically the band-pass filter to use `stopband=(5,25,'bandpass')`, and the output sample rate `out_fs=100`.
- `preprocessor_args` : dictionary of additional arguments to pass to the `preprocess.py` **offline** data pre-processor. Here we don't specify any.
- `model` : the machine learning model to use. Here it's the standard `cca` model.
- `clsfr_args` : dictionary of additional arguments to pass to the model fitting procedure `model_fitting.MultiCCA`. Here we specify the brain event types to model `evtlabs=('re','fe')` which means model the rising and falling edge responses only), the length of the brain response `tau_ms=450` which means a response of 450 milliseconds, and `rank=1` to fix the model rank of 1.

As it runs this script will generate a lot of text and then a final summary plot. The key things to understand are:

- **per-dataset decoding summary text**. For each dataset a lot of summary info is generated saying things like the number of trials in the data-set, the preprocessing, the classifier parameter settings used etc. At the end of all this will be a block of text like this:

```
IntLen 69 138 190 260 329 381 450 520 \n Perr 1.00 0.57 0.43 0.19 0.14 0.
05 0.05 0.05 AUDC 29.8 \n Perr(est) 0.87 0.49 0.48 0.11 0.07 0.07 0.05 0.03
PSAE 49.7 \n StopErr 1.00 1.00 1.00 0.60 0.43 0.33 0.33 0.33 AUSC 63.2 \n
StopThresh(P) 0.93 0.93 0.93 0.58 0.38 0.33 0.33 0.33 SSAE 8.7 \n
```

This is a textual summary of the models performance as a **decoding curve** which says how a particular performance measure changes as the system gets increasing amounts of data within a single trial. Thus, this allows you to see how you would expect the system to perform if you manually stopped the trial after, say, 500 samples (which @100hz = 5 seconds). There are 4 key metrics reported here to assess the system performance.

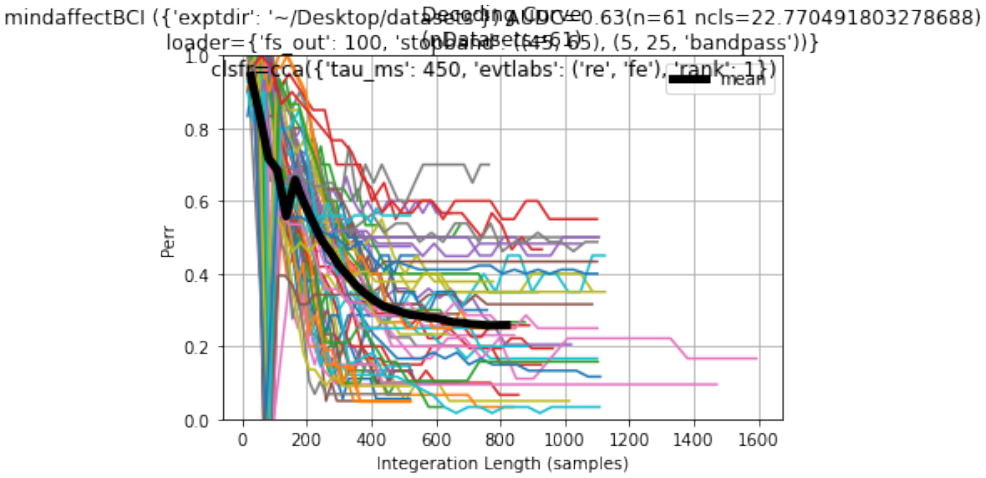
```
* **Perr** : this is the probability of an error -- basically this says that over all␣
 ↪of the training trials at each integeration-length if forced to stop what fraction␣
 ↪of the trails would be incorrect.
```

(continues on next page)

(continued from previous page)

```
* **Perr(est)** : in real usage the BCI doesn't actually know if it's prediction is␣
↪wrong, but has to estimate it's confidence itself.  As these predictions are used␣
↪to decide when to stop the trial, they should be as accurate as possible. ␣
↪Perr(est) represents over all the test trails the systems own estimate of it's␣
↪chance of being wrong with increasing integeration length.

* **StopErr** : The on-line system will Perr(est) to decide when to stop.  StopErr␣
↪gives an estimate of how accurate this method of stopping a trial can be.  Due to␣
↪trial-to-trial variation and accuracy of Perr(est) this can be better tha Perr (as␣
↪the system can stop early on easy trials, but later on hard trials), or it can be␣
↪much worse than Perr (if the estimates are very noisy so it stops at the wrong time.
↪)   So this estimate should be as low as possible as early as possible.

* **StopThresh** : When making the StopErr results the system estimates a Perr(est)␣
↪threshold which on-average stops at a given integeration length.  The StopThresh␣
↪line gives these thresholds.   Ideally, this should be **exactly** equal the␣
↪StopErr at every time point -- but if Perr(est) is incorrect or highly noisy it can␣
↪be higher or lower than desired.   This gives a guide to how reliable Perr(est)␣
↪stopping is for deciding when to stop a trial.

* **AUDC** : this is the Area Under the Decoding Curve, which is a single number --␣
↪lower better -- to characterise how fast and how low the Perr goes with time.

* **AUSC** : this is the Area Under the Stopping Curve, which is a similar single␣
↪performance number for the stopping curve.
* **PSAE** and **SSAE** : are Summed Average Error for the Perr(est) estimates, and␣
↪the StoppingThresholds.  Again lower is better for these.
```

- Dataset summary curve: After all the datasets have been run a final summary over all datasets will be generated, both as a textual summary like for the single trials. And as a datasets summary plot like this:



This plot shows for each dataset the summary Perr decoding curve, with the average over all datasets as the thick black line. Also in the title is a summary of the tested configuration and the cross datasets average AUDC to give an idea of the performance of this configuration over all the dataset.

[ ]:
```
analyse_datasets('mindaffectBCI',dataset_args=dict(exptdir=datasetsdir),
                 loader_args=dict(fs_out=100,stopband=((45,65),(5,25,'bandpass'))),
                 preprocess_args=None,
                 model='cca',test_idx=slice(10,None),clsfr_args=dict(tau_ms=450,
↪evtlabs=('re','fe'),rank=1))
```

(continues on next page)

(continued from previous page)

```
[ ]:
```

3.7.9 VideoTutorial: Adding Brain Controls to your Unity Game

Unity is a popular cross-platform video game development environment. In this video tutorial we show how you can easily add brain-controlled objects to your existing game using our unity API

<iframe width="560" height="315" src="https://www.youtube.com/embed/A-WC2KxZ9bM" frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture" allow-fullscreen></iframe>

**Note: this video is orginally for the mindaffect Kickstarter Kit so includes some references to the kit or fake-recogniser which are not relevant for the open-source release. All the unity specific parts remain unchanged. Just start the fake-recogniser with

3.7.10 VideoTutorial: Controlling a Phillips HUE with your brain

The Phillips HUE compute controlled lighting system. In this video tutorial we show how you can easily control the color of these lights with brain control using our python framework and the great phue python library.

**Note: this video is orginally for the mindaffect Kickstarter Kit so includes some references to the kit or fake-recogniser which are not relevant for the open-source release. All the unity specific parts remain unchanged. Just start the fake-recogniser with

3.7.11 Raspberry pi GPIO

Raspberry pi GPIO for Presentation

Mindaffect BCI is not limited to a screen based presentation. For example, physical lights and tactile vibrators can be used as means of stimulus. But first we need to connect our software to the desired hardware device (e.g. a lamp, LED, tactile vibrator). The software/hardware interfacing is done using GPIO. GPIO is a standard interface used to connect microcontrollers to other electronic devices. It can be used with sensors, diodes, displays, and System-on-Chip modules. For demonstration purposes we use a raspberry pi(Rpi) board. Any Rpi board with wireless connectivity is compatible with our software. In the rest of this section we discuss how to implement your physical presentation system of choice using Rpi's GPIO.

You will need

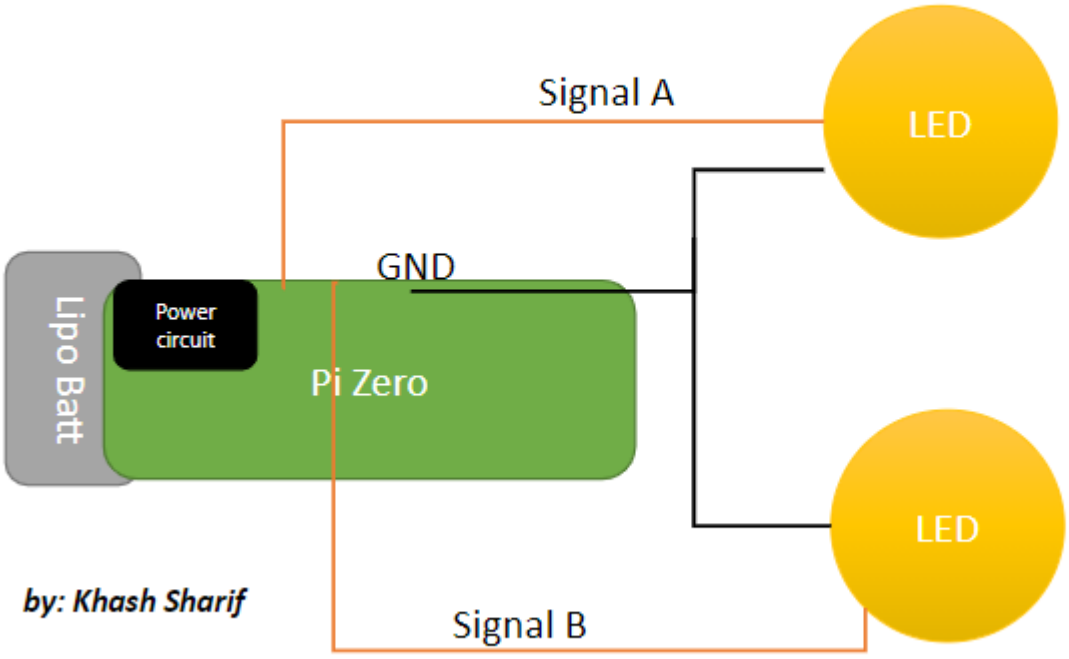
- Rpi zero W
- microSD card
- Lipo battery
- Power Circuit (DC-DC boost convertor)
- LED lights
- **Enclosure**
  - Case for the electronics

  - Light cover

- **Tools:**
  - Soldering equipment
  - Hot glue equipment

Design concept

The BCI stimulus is represented by binary codes. To transfer that into a physical flickering pattern the LED/lamp/tactile vibrator will be ON when the code value is 1 and will be toggled to OFF when the code is value is 0. For example: Code 1011010 is represented by ON-OFF-ON for twice as long-OFF-ON-OFF. A raspberry pi zero W is used for demonstration purposes. Below you can see a schematic diagram of how the electronic hardware looks like for a 2 button presentation module Where one LED plays the binary pattern from code A and the other one plays the pattern from code B. It is up to you to decide how many LEDs you would like to use.



Software setup

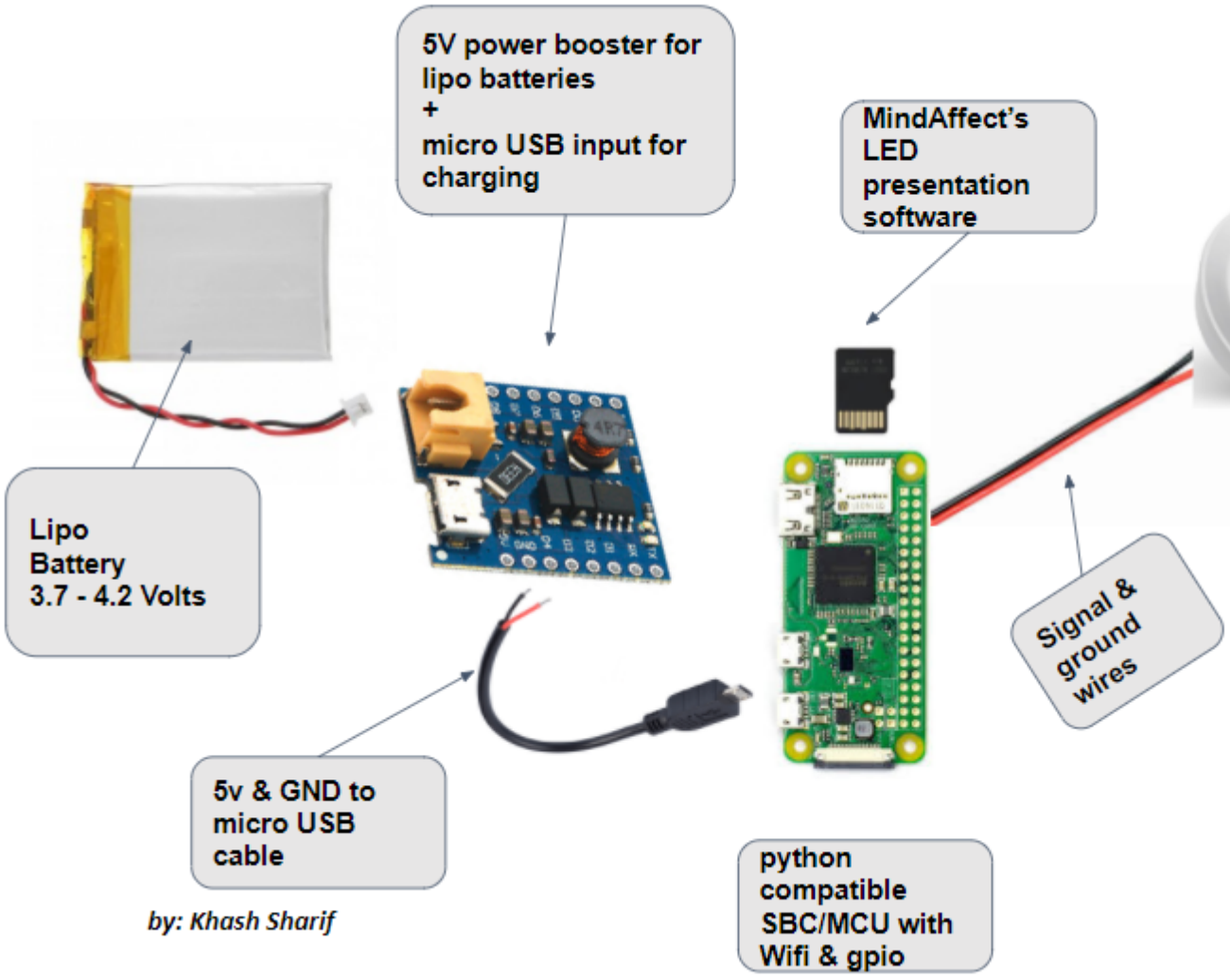
The board needs to be programmed with the MindAffect GPIO software build on top of gpizero library. To do so, follow the steps below.

1. Follow this guide to install the Raspbian OS and enable SSH on your Rpi.

2. **Now, connect to the raspberry pi via SSH and run the following command in the terminal** sudo apt-get install git default-jre default-jdk python3 build-essential cli-common xterm ant gradle
3. **Clone the pymindAffectBCI repository by using the following command** git clone https://github.com/mindaffect/pymindaffectBCI
4. **Install the necessary bits to your local python path:**
    1. change to the directory where you cloned the repository.
    2. **Add this module to the python path, and install dependencies** python3 -m pip install -e .
5. **If you would like to configure the Rpi such that the MindAffect GPIO software automatically starts on boot, do the followi**
    1. **Open the autostart script by running the following command**
        1. Sudo nano /etc/xdg/lxsession/LXDE-pi/autostart
        2. **Add the following line to the autostart file** python3 -m mindaffect-BCI.examples.presentation.rpigpio
        3. Press ctrl+x, then press Y and then Enter to save changes.
        4. The changes will take effect after a reboot.
6. Make sure you are connected to the same WiFi network that is used by the machine running the MindAffectBCI software.
7. The rpigpio python script uses GPIO pins 2,3,4 by default. You can customize it to your liking.
8. Allright, the software is set up. Next, the hardware needs to be assembled.

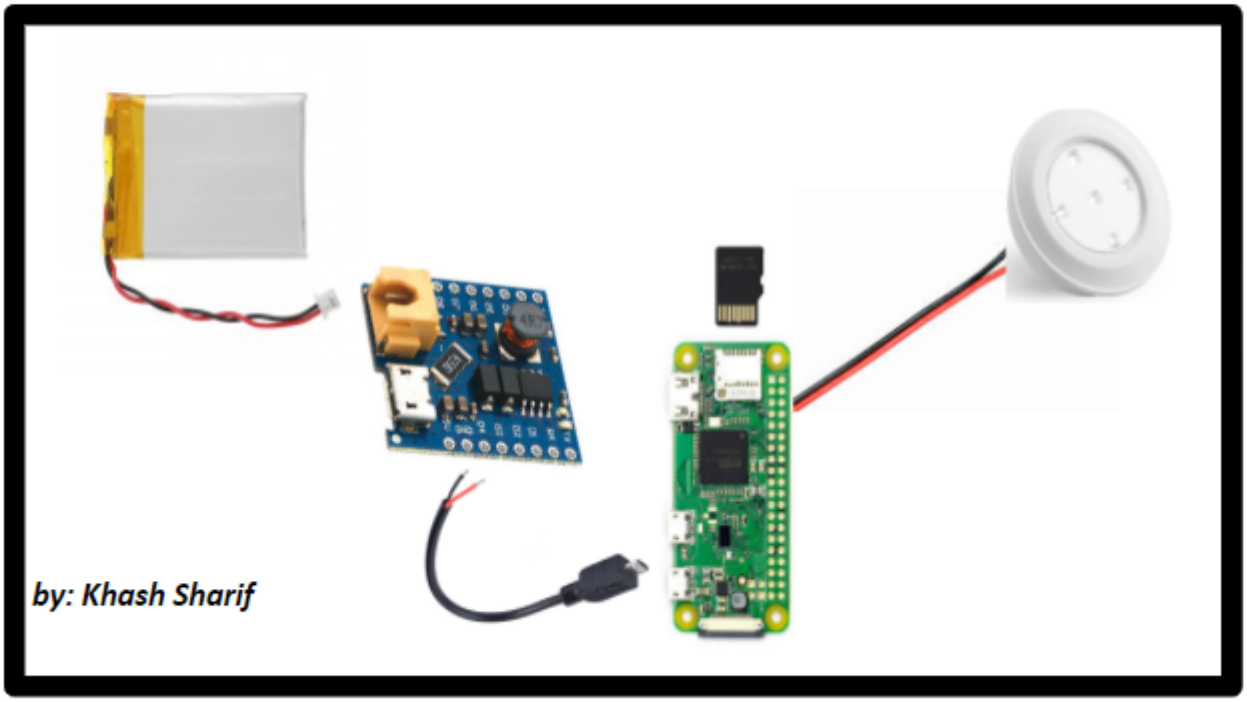
Directions for setting up a minimum presentation hardware

1. **Connect the + pins of the LEDs to the GPIO pins**
    1. By default pins 2,3,4 are used in the rpgpio.py script. You don't have to use all the pins, but you need to inform the software about How many pins are going to be used by the LEDs
2. Connect the - pins of the LEDS to the GND GPIO pin
3. **solder the micro USB cable to the power circuit pins**
    - red wire –> 5v pin
    - black wire –> GND pin
4. Connect the power circuit to the Rpi board.



Caution If you are not familiar with safety cautions related to LiPo batteries use other safe batteries (such as AA battery packs) or learn how to safely use a LiPO battery. The power circuit & the battery can blow up if you plug in the battery the wrong way. Use at your own risk or choose a safe battery instead.

5. As soon as the battery is connected to the power circuit the Rpi board will boot
6. **To shutdown the system, you can connect to the Rpi board via SSH and run the following command:**
    1. Sudo shutdown -h now
    2. Then you can safely unplug the battery after the RPi's green lights turn off
7. Place all the assembled parts in the enclosure and insulator the electronics using hot glue.



The final LED button looks like this:

To run the full demo

1. **On your host computer, go to the directory of pymindaffectBCI/mindaffectBCI . Open the online_bci.json file and set the**

    - "presentation":"None"

2. Now plug in the battery to the power circuit of the LED button and close the enclosure.
3. Make sure your host computer and the LED button are connected to the same network
4. **The LED button should connect to the host computer and the presentation starts automatically and runs using the default**

    - Number of calibration and prediction trials
    - Number of LEDs
    - The LED to GPIO pin mapping
    - Speed of stimulus in Hz

```
71  def init(framerate_hz=15, numleds=2, led2gpiopin=(2,3
72      """setup the pi for GPIO based presentation
73
74      Args:
75          framerate_hz (float, optional): framerate for
76          numleds (int, optional): number of leds to fl
77          led2gpiopin (tuple, optional): the LED index
78          nCal (int, optional): number of calibration t
79          nPred (int, optional): number of prediction t
80      """
```

Here's how a one button LED presentation looks like (the LED button stays ON for a while when it is selected by the user's brain response)

And a two button system looks like

RPI GPIO for control

You can control a physical device using MindAffect's output module and a board with GPIO. For demonstration purposes we use GPIO pins of a raspberry pi board to control other physical devices.

3.7.12 Add a new amplifier to mindaffectBCI

Out of the box mindaffectBCI supports a large range of amplifiers, either via. it's use of brainflow or with amplifier drivers developed by MindAffect. But what if you have a new wizzy amplifier and it's not currently supported by brainflow? How can you easily add support for this cool new device to the mindaffectBCI – ideally without introducing a lot of extra dependencies in your code?

In this tutorial you will learn:

1. The low-level format used to stream to the mindaffectBCI hub
2. The importance of **device-level** time-stamps in ensuring the data you send it as good as possible for BCI applications
3. How to write a simple 'fake-data' simulated amplifier stream in Python.

The mindaffectBCI DATAPACKET transmission format

Data is sent from the amplifier to the mindaffectBCI in packets containing an array of channels by samples in 32-bit floats (were

1. Open a TCP network socket to connect to the hub on port 8400
2. Send a stream of DATAPACKET to the hub

Note: At it's lowest level this is a **one-way** data stream – so your amp driver logic can be extreemly simple. Further, you do not **need** to provide any meta-information about the amplifier, e.g. model, a sample rate, or channel names etc., for it to work (though it is nice if you can send this information in a DATAHEADER packet.) Thus, quickly developing a new amp driver for testing can be extreemly simple – as you will see below it's literially ~6 lines of pure-python.

The detailed format of the DATAPACKET messages (along with all the other message types used by the mindaffectBCI) are given in the system message specification. The core section for the DATAPACKET messages is repeated here for clarity.

Based on this format, in python given an integer *timestamp*, raw data in *samples* which is a (samples,channels) *np.float32* numpy array and using the *struct* package, you can make a valid datapacket with:

```
DP = struct.pack("<BBHii%df"%(samples.size),ord('D'),0,4+4+samples.size*4,timestamp,
↪samples.shape[-1],*(s for s in samples.ravel()))
```

Note: This line uses some horrible python hacks; like: *ord('D')* to convert char->integer, *samples.ravel()* to convert the n-d samples to a 1-d matrix, *(s for s in samples.ravel())* to convert the nd-array to a python tuple, and the finally **(...)* to expand the tuple into a set of arguments.

Minimal Acquisation Driver : Python

Note: this example designed for exposition purposes, implementators are better adviced to use the *utopiaclient.py* API, as it provides a more complete interface, with e.g. auto-discovery, error-recovery, two-way communication, and access to the full message vocabularly.

To make the absolute minimum *fake-data* streamer we need to do 5 things:

1. Open a TCP socket to connect to the hub.:

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.open('localhost',8400)
```

2. Get the fake-data packet:

```
n_ch = 4
n_samples = 10
samples = np.random.standard_normal((n_ch,n_samples)).astype(np.float32)
```

3. Get the current time-stamp:

```
timestamp = int(time.perf_counter()*1000) % (1<<31) # N.B. MUST fit in 32bit␣
↪int
```

4. Make the DATAPACKET:

```
DP = struct.pack("<BBHii%df"%(samples.size),ord('D'),0,4+4+samples.size*4,
↪timestamp,samples.shape[-1],*(s for s in samples.ravel()))
```

5. send the message:

```
sock.send(DP)
```

Or to wrap it all up into a single 10-line code block (without imports), with a loop to stream for-ever, and a sleep to rate-limit to a desired effective sample rate:

```
import numpy as np
import time
import socket
import struct

def fakedata_stream(host='localhost', sample_rate=100, n_ch=4, packet_samples=10):
    inter_packet_interval = packet_samples / sample_rate

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((host,8400))

    while True:
        samples = np.random.standard_normal((n_ch,packet_samples)).astype(np.
↪float32)
        timestamp = int(time.perf_counter()*1000) % (1<<31) # N.B. MUST fit in␣
↪32bit int
        DP = struct.pack("<BBHii%df"%(samples.size),ord('D'),0,4+4+samples.size*4,
↪timestamp,samples.shape[-1],*(s for s in samples.ravel()))
        sock.send(DP)
        time.sleep(inter_packet_interval) # sleep to rate limit to sample_rate Hz
```

Congratulations, you have just written your own custom datapacket streamer for the mindaffect BCI.

To adapt this to use data from an actual hardware device, then simply replace the *samples = np.random.standard_normal...* line with a call to the hardware function which gets the actual samples from the amplifier.

The Importance of Amplifier timestamps

At it's core any evoked-response BCI (like the mindaffect BCI) must align at least two data-streams, namely the EEG stream (from the amplifier) and the STIMULUS stream (from the presentation device). Doing this alignment with high latency links (such as wireless network connections) can be a complex problem. The solution used in the mindaffect BCI is to use a **local** clock on the device (i.e. amplifier, screen) to attach accurate **timestamps** to the data at source, and then use a jitter rejecting and step detection algorithm in the decoder to align the time-stamp streams (which due to electronic issues can have different offsets and may drift relative to each other) to the common decoder clock.

What this means for amplifier implementors is that **it is very important** to time-stamp your data as close to the source as possible. We have found that using the poor quality clocks in a cheap devices is a better time-stamp source than an high quality clock in a PC – basically because even a poor quality device clock has a sub-millisecond jitter and only drifts by approx 1 millisecond / second, whereas wireless transmission jitter can be 10 to 100 milliseconds / second with a similar 1ms/s drift. When coupled to potential sample loss in transmission, this makes 'recieve-time' timestamps a poor subistute for 'measurment-time' device-level timestamps.

Summary

Adding a new amplifier to the mindaffect BCI can be done by either:

1. Adding the new amplifier to brainflow
2. Streaming the data on a TCP socket in the timestamped DATAPACKET format

3.8 Going Further : BCI-types, Decoder Config

You can run the BCI in different modes by specifying different arguments on the command line. Or by modifying the basic configuration file online_bci.json.

3.8.1 Alternative BCI types / Stimulus

By default we use the mindaffect NoiseTagging style stimulus with a 25-symbol letter matrix for presentation. You can easily try

- _symbols_ : can either by a list-of-lists of the actual text to show, for example for a 2x2 grid of sentences.

```
"presentation_args":{
    "symbols":[ ["I'm happy","I'm sad"], ["I want to play","I want to sleep"] ],
    "stimfile":"mgold_65_6532_psk_60hz.png",
    "framesperbit":1
}
```

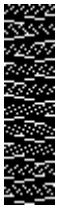
  or a file from which to load the set of symbols as a *comma-separated* list of strings like the file symbols.txt.

- _stimfile_ : is a file which contains the stimulus-code to display. This can either be a text-file with a matrix specified with a white-space separated line per output or a png with the stimulus with outputs in 'x' and time in 'y' like.

You can clearly see the difference between the two main types of BCI stimulus file used here when viewed as an image. Firstly, this is the stimulus file for the noisecodes.

which clearly shows the noise-like character of this code.

By contrast the, classical P300 row-column speller stimulus sequence looks like.



which shows the more structured row-column structure, and that only a few outputs are 'on' at any time.

3.8.2 Change Decoder parameters

The decoder is the core of the BCI at it takes in the raw EEG and stimulus information and generates predictions about which stimulus the user is attending to. Generating these predictions relies on signal processing and machine learning techniques to learn the best decoding parameters for each user. However, ensuring best performance means the settings for the decoder should be appropriate for the particular BCI being used. The default decoder parameters are found in the configuration file online_bci.json in the *decoder_args* section, and are setup for a noisetagging BCI.

The default settings for noisetagging are

```
"decoder_args":{
    "stopband" : [3,25,"bandpass"],
    "out_fs" : 80,
    "evtlabs" : ["re","fe"],
    "tau_ms" : 450,
    "calplots" : true,
    "predplots" : false
}
```

The key parameters here are:

- *stopband*: this is a temporal filter which is applied as a pre-processing step to the incomming data. This is important to remove external noise so the decoder can focus on the target brain signals. Here the filter is specified as a list of bandpass or band stop filters, which specify which signal frequencies should be suppressed, (where, in classic python fashion -1 indicates the max-possible frequency). Thus, in this example only frequencies between 3 and 25Hz remain after filtering.
- *out_fs*: this specifies the post-filtering sampling rate of the data. This reduces the amount of data which will be processed by the rest of the decoder. Thus, in this example after filtering the data is re-sampled to 80Hz. (Note: to avoid []() out_fs should be greater than 2x the maximum frequency passed by the stop-band).
- *evtlabs*: this specifies the stimulus properties (or event labels) the decoder will try to predict from the brain responses. The input to the decoder (and the brain) is the raw-stimulus intensity (i.e. it's brightness, or loudness).

However, depending on the task the user is performing, the brain may *not* respond directly to the brightness, but some other property of the stimulus. For example, in the classic P300 'odd-ball' BCI, the brain responds not to the raw intensity, but to the start of *surprising* stimuli. The design of the P300 matrix-speller BCI means this response happens when the users choosen output 'flashes', or gets bright. Thus, in the P300 BCI the brain responses to the rising-edge of the stimulus intensity. Knowing, exactly what stimulus property the brain is responding to is a well studied neuroscientific research question, with examples including, stimulus-onset (a.k.a. rising-edge, or 're'), stimulus-offset (a.k.a. falling-edge, or 'fe'), stimulus intensity ('flash'), stimulus-duration etc. Getting the right stimulus-coding is critical for BCI peformance, see stim2event.py for more information on supported event types.

- *tau_ms*: this specifies the maximum duration of the expected brain response to a triggering event in *milliseconds*. As with the trigger type, the length of the brian response to a triggering event depends on the type of response expected. For example for the P300 the response is between 300 and 600 ms after the trigger, whereas for a VEP the response is between 100 and 400 ms. Ideally, the response window should be as small as possible, so the learning system only gets the brain response, and not a lot of non-response containing noise which could lead the machine learning component to overfit.

3.9 Project Ideas / Inspiration

To inspire you with some things that you can build and brain control, checkout our demo videos:

- Brain Control in Untity
- Lego BOOST laser control
- Philips Hue control
- Sphero control
- VR gaming
- Space Invaders
- Speller iPhone
- Tin Throwing
- Brain Controlled TV Remote
- Speller in Python

Other media: - National Geographic feature - Helping ALS patient - TechCrunch Berlin

3.10 Frequently Asked Questions

3.10.1 How do I improve my calibration accuracy?

First, open the signal viewer by pressing 0 in the main menu to check the quality of the EEG signal. If the reported noise to signal ratio is high, try to reduce the noise as follows:

- Make sure your headset is properly set up and fitted. See our instructions here: *Set-up & fitting of MindAffect water electrodes*.
- Move away from wall outlets, plugged in electronic devices, and other potential sources of line-noise.
- If possible, place the amplifier behind you so your body is between the machine running the mindaffcetBCI and the amplifier.

- Place the EEG hardware close to or on your body and run the electrode cables over your back.
- Check all the connections between the electrodes in your headset and the amplifier.

Second, make sure that you have followed our OS Optimization instructions: *OS Optimization*, and you are running the BCI in fullscreen mode: *How do I run the BCI in full-screen mode?*. If the issue still remains you may have to dive deeper in the timing stability of your system. To do this we provide tutorials on:

- *How to build your own optical sensor*
- Building a trigger circuit
- triggercheckRef Analysing your opto and trigger data

3.10.2 My calibration accuracy is fine but prediction mode does not work.

- Add more calibration trials to the model by running the calibration sequence multiple times, or change the `ncal` argument in the `online_bci.json` configuration file (or the config file you are currently using).
- Run the BCI in always full-screen mode: *How do I run the BCI in full-screen mode?*
- Inconsistent frame timing between calibration and prediction can cause this issue. To check the frametime stability of your system follow our tutorials on:
  - *How to build your own optical sensor*
  - Building a trigger circuit
  - Analysing your opto and trigger data

3.10.3 I'm getting an `Acq did not start correctly` and/or a `brainflow.board_shim.BrainFlowError` message.

- Check if your amplifier is turned on and –if needed– the usb-dongle is plugged in.
- When using an OpenBCI amplifier with the WIFI-shield, make sure it is connected to the same wireless network as the machine running the mindaffcetBCI.
- Check you serial port settings as described in the COMref section of the installation instructions.

3.10.4 How do I run the BCI in full-screen mode?

To run the BCI in full-screen mode set the `fullscreen` parameter in the `onlin_bci.json` configuration file –or in any other `.json` config file you are currently using– to `true`.

3.10.5 Can I use the mindaffectBCI without an EEG acquisition device?

In some scenarios it is useful to run the BCI without having to connect an amplifier that's streaming real brain data (e.g. debugging/developing other components of the BCI). To run the full BCI stack (i.e. hub, acquisition, decoder and presentation) with a fake data stream, launch it with the `debug.json` config file:

```
python -m mindaffectBCI.online_bci --config_file debug.json
```

Alternatively, do run full decoder stack (i.e. hub, acquisation and decoder) *without* presenation use:

```
python -m mindaffectBCI.online_bci --config_file fake_recogniser.json
```

When using the fake data stream, calibration and cued prediction performance will be 100%. In Free Typing mode selections are made randomly.

3.10.6 I'm getting a *framework not found* error on Mac OS

On mac-os Big-Sur there is a known issue with older versions of pyglet see here. The solution is to ensure you are running pyglet 1.5.11 or higher. You can directly update your pyglet install with: `pip3 install --upgrade pyglet`

3.10.7 I'm getting poor performance on linux

As we are a small team, we have decided to focus our development and testing effort mainly on Windows PCs. We have tested the BCI on linux and mac-os, and it technically works. However, as mentioned here triggercheckRef it is also important that the screen redraws be accuratly time-stamped. In our testing on linux this time-locking was less robust than with an optimized windows installation. We believe this is can be addressed by a correct graphics system configuration, but have not identified it as yet. We would welcome feedback from the community about how to setup linux better.

3.11 Python API

3.11.1 mindaffectBCI package

Subpackages

mindaffectBCI.decoder package

Subpackages

mindaffectBCI.decoder.offline package

Submodules

mindaffectBCI.decoder.offline.load_brainsonfire module

mindaffectBCI.decoder.offline.load_brainsonfire.**load_brainsonfire**(*datadir*, *sessdir=None*, *sessfn=None*, *fs_out=60*, *stopband=((45, 65), (0, 1), (25, -1))*, *subtriallen=10*, *nvirt=20*, *chIdx=slice(None, 64, None)*, *verb=2*)

mindaffectBCI.decoder.offline.load_brainsonfire.**testcase**()

mindaffectBCI.decoder.offline.load_brainstream module

mindaffectBCI.decoder.offline.load_brainstream.**load_brainstream**(*datadir*, *sessdir=None*, *sessfn=None*, *fs_out=60*, *ifs=None*, *fr=None*, *stopband=((45, 65), (5, 25, 'bandpass'))*, *verb=0*, *ch_names=None*)

Load and pre-process a brainstream offline save-file and return the EEG data, and stimulus information

Parameters

- **datadir** (*str*) – root of the data directory tree
- **sessdir** (*str, optional*) – sub-directory for the session to load. Defaults to None.
- **sessfn** (*str, optional*) – filename for the session information. Defaults to None.
- **fs_out** (*float, optional*) – [description]. Defaults to 100.
- **ifs** (*float, optional*) – the input data sample rate.
- **fr** (*float, optional*) – the input stimulus frame rate.
- **stopband** (*tuple, optional*) – Specification for a (cascade of) temporal (IIR) filters, in the format used by *mindaffectBCI.decoder.utils.butter_sosfilt*. Defaults to ((45,65),(5.5,25,'bandpass')).
- **ch_names** (*tuple, optional*) – Names for the channels of the EEG data.

Returns

the pre-processed per-trial EEG data Y (np.ndarray (nTrl,nSamp,nY)): the up-sampled stimulus information for each output coords (list-of-dicts (3,)): dictionary with meta-info for each dimension of X & Y. As a minimum this contains

"name"- name of the dimension, "unit" - the unit of measurment, "coords" - the 'value' of each element along this dimension

Return type X (np.ndarray (nTrl,nSamp,nCh))

mindaffectBCI.decoder.offline.load_brainstream.**testcase**()

mindaffectBCI.decoder.offline.load_cocktail module

mindaffectBCI.decoder.offline.load_cocktail.**argsort**(*l*)

mindaffectBCI.decoder.offline.load_cocktail.**load_cocktail**(*datadir*, *sessdir=None*, *sessfn=None*, *fs_out=60*, *stopband=((45, 65), (0, 5), (25, -1))*, *verb=0*, *trlen_ms=None*, *subtrialen=10*)

mindaffectBCI.decoder.offline.load_cocktail.**squeeze**(*v*)

mindaffectBCI.decoder.offline.load_cocktail.**testcase**()

mindaffectBCI.decoder.offline.load_mTRF_audio module

mindaffectBCI.decoder.offline.load_mTRF_audio.**load_mTRF_audio**(*datadir*, *regressor='envelope'*, *ntrl=15*, *stopband=((45, 65), (0, 0.5), (15, -1))*, *fs_out=60*, *nvirt_out=30*, *verb=1*)

mindaffectBCI.decoder.offline.load_mark_EMG module

mindaffectBCI.decoder.offline.load_mark_EMG.**load_mark_EMG**(*datadir*, *sessdir=None*, *sessfn=None*, *fs_out=60*, *stopband=((45, 65), (0, 10), (45, 55), (95, 105), (145, -1))*, *filterbank=None*, *verb=0*, *log=True*, *whiten=True*, *plot=False*)

mindaffectBCI.decoder.offline.load_mark_EMG.**testcase**()

mindaffectBCI.decoder.offline.load_mindaffectBCI module

mindaffectBCI.decoder.offline.load_mindaffectBCI.**load_mindaffectBCI**(*source*, *datadir: str = None*, *sessdir: str = None*, *fs_out: float = 100*, *stopband=((45, 65), (5.5, 25, 'bandpass'))*, *order: int = 6*, *ftype: str = 'butter'*, *verb: int = 0*, *iti_ms: float = 1000*, *trlen_ms: float = None*, *offset_ms: float = (-500, 500)*, *ch_names=None*, ***kwargs*)

Load and pre-process a mindaffectBCI offline save-file and return the EEG data, and stimulus information

Parameters

- **source** (*str, stream*) – the source to load the data from, use '-' to load the most recent file from the logs directory.
- **fs_out** (*float, optional*) – [description]. Defaults to 100.
- **stopband** (*tuple, optional*) – Specification for a (cascade of) temporal (IIR) filters, in the format used by *mindaffectBCI.decoder.utils.butter_sosfilt*. Defaults to ((45,65),(5.5,25,'bandpass')).
- **order** (*int, optional*) – the order of the temporal filter. Defaults to 6.
- **ftype** (*str, optional*) – The type of filter design to use. One of: 'butter', 'bessel'. Defaults to 'butter'.
- **verb** (*int, optional*) – General verbosity/logging level. Defaults to 0.

- **iti_ms** (*int, optional*) – Inter-trial interval. Used to detect trial-transitions when gap between stimuli is greater than this duration. Defaults to 1000.
- **trlen_ms** (*float, optional*) – Trial duration in milli-seconds. If None then this is deduced from the stimulus information. Defaults to None.
- **offset_ms** (*tuple, (2,) optional*) – Offset in milliseconds from the trial start/end for the returned data such that X has range [tr_start+offset_ms[0] -> tr_end+offset_ms[0]]. Defaults to (-500,500).
- **ch_names** (*tuple, optional*) – Names for the channels of the EEG data.

Returns

the pre-processed per-trial EEG data Y (np.ndarray (nTrl,nSamp,nY)): the up-sampled stimulus information for each output coords (list-of-dicts (3,)): dictionary with meta-info for each dimension of X & Y. As a minimum this contains

"name"- name of the dimension, "unit" - the unit of measurment, "coords" - the 'value' of each element along this dimension

Return type X (np.ndarray (nTrl,nSamp,nCh))

mindaffectBCI.decoder.offline.load_mindaffectBCI.**testcase**()

mindaffectBCI.decoder.offline.load_ninapro_db2 module

mindaffectBCI.decoder.offline.load_ninapro_db2.**load_ninapro_db2**(*datadir*, *stopband=((0, 15), (45, 65), (95, 125), (250, -1))*, *envelopeband=(10, -1)*, *trlen_ms=None*, *fs_out=60*, *nvirt=20*, *rectify=True*, *whiten=True*, *log=True*, *plot=False*, *filterbank=None*, *zscore_y=True*, *verb=1*)

mindaffectBCI.decoder.offline.load_ninapro_db2.**testcase**()

mindaffectBCI.decoder.offline.load_openBMI module

mindaffectBCI.decoder.offline.load_openBMI.**get_trl_ep_idx**(*trl_idx*, *iti*)

convert set stimulus times into trials and epochs

mindaffectBCI.decoder.offline.load_openBMI.**load_openBMI**(*datadir*, *sessdir=None*, *sessfn=None*, *fs_out=60*, *stopband=((45, 65), (0, 1), (25, -1))*, *CAR=False*, *verb=1*, *trlen_ms=None*, *offset_ms=(0, 0)*, *ppMI=True*, *ch_names=None*)

Load and pre-process a openBMI <https://academic.oup.com/gigascience/article/8/5/giz002/5304369> offline save-file and return the EEG data, and stimulus information

Parameters

- **datadir** (*str*) – root of the data directory tree
- **sessdir** (*str, optional*) – sub-directory for the session to load. Defaults to None.
- **sessfn** (*str, optional*) – filename for the session information. Defaults to None.
- **fs_out** (*float, optional*) – [description]. Defaults to 100.
- **stopband** (*tuple, optional*) – Specification for a (cascade of) temporal (IIR) filters, in the format used by *mindaffectBCI.decoder.utils.butter_sosfilt*. Defaults to ((45,65),(5.5,25,'bandpass')).
- **trlen_ms** (*float, optional*) – Trial duration in milli-seconds. If None then this is deduced from the stimulus information. Defaults to None.
- **offset_ms** (*tuple, (2,) optional*) – Offset in milliseconds from the trial start/end for the returned data such that X has range [tr_start+offset_ms[0] -> tr_end+offset_ms[0]]. Defaults to (-500,500).
- **ch_names** (*tuple, optional*) – Names for the channels of the EEG data.
- **CAR** (*bool*) – flag if we should common-average-reference the raw EEG data

Returns

the pre-processed per-trial EEG data Y (np.ndarray (nTrl,nSamp,nY)): the up-sampled stimulus information for each output coords (list-of-dicts (3,)): dictionary with meta-info for each dimension of X & Y. As a minimum this contains

"name"- name of the dimension, "unit" - the unit of measurment, "coords" - the 'value' of each element along this dimension

Return type X (np.ndarray (nTrl,nSamp,nCh))

mindaffectBCI.decoder.offline.load_openBMI.**make_ssvep_ref_signals**(*nsample*, *freqs*, *phases=None*, *fs=1*, *binarize=True*)

make periodic reference signals

mindaffectBCI.decoder.offline.load_openBMI.**testcase**()

mindaffectBCI.decoder.offline.load_p300_prn module

mindaffectBCI.decoder.offline.load_p300_prn.**load_p300_prn**(*datadir*, *sessdir=None*, *sessfn=None*, *fs_out=60*, *offset_ms=(-1000, 1000)*, *ifs=None*, *fr=None*, *stopband=((45, 65), (1, 25, 'bandpass'))*, *order=6*, *subtriallen=10*, *verb=0*, *nvirt=20*, *chidx=slice(None, 64, None)*)

mindaffectBCI.decoder.offline.load_p300_prn.**testcase**()

mindaffectBCI.decoder.offline.load_twofinger module

mindaffectBCI.decoder.offline.load_twofinger.**load_twofinger**(*datadir*, *sessdir=None*, *sessfn=None*, *fs_out=60*, *stopband=((45, 65), (0, 1), (25, -1))*, *subtriallen=10*, *nvirt=20*, *verb=0*, *ch_idx=slice(None, 32, None)*)

mindaffectBCI.decoder.offline.load_twofinger.**testcase**()

mindaffectBCI.decoder.offline.read_buffer_offline module

class mindaffectBCI.decoder.offline.read_buffer_offline.**ftevent**(*s*, *t*, *v*, *o=None*, *d=None*)

Bases: object

class mindaffectBCI.decoder.offline.read_buffer_offline.**ftheader**(*nch*, *nsamp*, *nevt*, *fs*, *data_type*, *labels=None*)

Bases: object

mindaffectBCI.decoder.offline.read_buffer_offline.**read_buffer_offline_data**(*f*, *hdr*)

mindaffectBCI.decoder.offline.read_buffer_offline.**read_buffer_offline_events**(*f*)

mindaffectBCI.decoder.offline.read_buffer_offline.**read_buffer_offline_header**(*f*)

mindaffectBCI.decoder.offline.read_buffer_offline.**testcase**()

mindaffectBCI.decoder.offline.read_mindaffectBCI module

mindaffectBCI.decoder.offline.read_mindaffectBCI.**datapackets2array**(*msgs*, *sample2timestamp='lower_bound_trac*, *timestamp_ch=None*)

Convert a set of datapacket messages to a 2-d numpy array (with timestamp channel)

Parameters

- **msgs** (*[type]*) – list of DataPacket messages
- **sample2timestamp** (*str, optional*) – filtering function, to filter the data-packet time-stamps using the increasing sample counts. Defaults to 'lower_bound_tracker'.
- **timestamp_ch** (*int, optional*) – If set, channel which contains the timestamp information. Defaults to None.

Returns the extracted samples in a numpy array

Return type X((t,d) np.ndarray)

mindaffectBCI.decoder.offline.read_mindaffectBCI.**read_DataHeader**(*line: str*)

read a data-header line from a mindaffectBCI offline save file

Parameters **line** (*str*) – the line to read

Returns a mindaffectBCI.utopiaclient.messages.DataPacket object containing (nsamp,d) EEG data

Return type *DataPacket*

mindaffectBCI.decoder.offline.read_mindaffectBCI.**read_DataPacket**(*line: str*)

read a data-packet line from a mindaffectBCI offline save file

Parameters **line** (*str*) – the line to read

Returns a mindaffectBCI.utopiaclient.messages.DataPacket object containing (nsamp,d) EEG data

Return type *DataPacket*

mindaffectBCI.decoder.offline.read_mindaffectBCI.**read_ModeChange**(*line: str*)

read a mode-change line from a mindaffectBCI offline save file

Parameters **line** (*str*) – the line to read

Returns a mode-change object, with the new mode information

Return type *ModeChange*

mindaffectBCI.decoder.offline.read_mindaffectBCI.**read_NewTarget**(*line: str*)

read a newtarget message line from a mindaffectBCI offline save file

Parameters **line** (*str*) – the line to read

Returns a newtarget object

Return type *NewTarget*

mindaffectBCI.decoder.offline.read_mindaffectBCI.**read_Selection**(*line: str*)

read a Selection message line from a mindaffectBCI offline save file

Parameters **line** (*str*) – the line to read

Returns a selection object - Note: currently the selection information is *not* valid

Return type *Selection*

mindaffectBCI.decoder.offline.read_mindaffectBCI.**read_StimulusEvent**(*line: str*)

read a stimulus event message from a line of a mindaffectBCI offline save file

mindaffectBCI.decoder.offline.read_mindaffectBCI.**read_clientip**(*line: str*)

read the client-ip-address from a message line from a mindaffectBCI offline save file

Parameters **line** (*str*) – the line to read

Returns the client ip-address

Return type ip (str)

mindaffectBCI.decoder.offline.read_mindaffectBCI.**read_clientts**(*line: str*)

read the client-timestamp from a message line from a mindaffectBCI offline save file

Note: the client-timestamp is the *raw* timestamp sent by the client, in the clients local clock.

Parameters **line** (*str*) – the line to read

Returns the client timestamp

Return type clientts (int)

mindaffectBCI.decoder.offline.read_mindaffectBCI.**read_mindaffectBCI_data_messages**(*source*, *regress=Fals*
times-
*tamp_wrap_*
***kwargs*)

read an offline mindaffectBCI save file, and return raw-data (as a np.ndarray) and messages.

Parameters

- **source** (*str*) – the file name to load the data from
- **regress** (*bool, optional*) – How to map from client-specific to a common time-stamp basis. Defaults to False.
- **timestamp_wrap_size** (*tuple, optional*) – The bit-resolution of the time-stamps. Defaults to (1<<24).

Returns the time-stamped data stream messages (list messages): the (non-datapacket) messages in the file

Return type data (np.ndarray (nsamp,d) float)

mindaffectBCI.decoder.offline.read_mindaffectBCI.**read_mindaffectBCI_message**(*line: str*)

Read a mindaffectBCI message from a line of text

Parameters **line** (*str*) – A line containing a mindaffectBCI message

Returns The decoded message as a message class.

Return type message_type

mindaffectBCI.decoder.offline.read_mindaffectBCI.**read_mindaffectBCI_messages**(*source*, *regress: bool = False*)

read all the messages from a mindaffetBCI offline save file

Parameters

- **source** (*[str, stream]*) – the log file messages source, can be file-name, or IO-stream, or string
- **regress** (*bool, optional*) – How should we regress the client-time stamps onto the server time-stamps. If False then use the server-time-stamps, if None then leave the client-time-stamps, if True then use robust-regression to map from client to server time-stamps.
- **to False.** (*Defaults*) –

Returns a list of all the decoded messages

Return type (list, messages)

mindaffectBCI.decoder.offline.read_mindaffectBCI.**read_recievedts**(*line: str*)
read the recieved-timestamp from a message line from a mindaffectBCI offline save file

Note: the recieved-timestamp is the time the client-message was recieved, measured on the servers clock.

Parameters **line** (*str*) – the line to read

Returns the timestamp

Return type ts (int)

mindaffectBCI.decoder.offline.read_mindaffectBCI.**read_serverts**(*line: str*)
read the server-timestamp from a message line from a mindaffectBCI offline save file

Note: the server-timestamp is the client-timestamp after mapping to the servers local clock. Thus, server timestamps are directly comparable for all clients.

Parameters **line** (*str*) – the line to read

Returns the server timestamp

Return type ts (int)

mindaffectBCI.decoder.offline.read_mindaffectBCI.**rewrite_timestamps2servertimestamps**(*msgs*)
rewrite message client-timestamps to best-fit server time-stamps

mindaffectBCI.decoder.offline.read_mindaffectBCI.**robust_timestamp_regression**(*x*, *y*)
Given 2 time-stamp streams, e.g. one from server, one from client, compute a robust, outlier resistant linear mapping from one to the other.

Parameters

- **x** (*int/float, (ntimes,)*) – the source time-stamps
- **y** (*int/float, (ntimes,)*) – the destination time-stamps

Returns the gain and bias for the linear map such that: y = ab[0]*x + ab[1]

Return type ab

mindaffectBCI.decoder.offline.read_mindaffectBCI.**testcase**(*fn=None*)
testcase, load reference datafile

Module contents

Submodules

mindaffectBCI.decoder.FileProxyHub module

class mindaffectBCI.decoder.FileProxyHub.**FileProxyHub**(*filename: str = None*, *speedup: float = None*, *use_server_ts: bool = True*)

Bases: object

Proxy UtopiaClient which gets messages from a saved log file

autoconnect(**args*, ***kwargs*)
[summary]

getNewMessages(*timeout_ms*)
[summary]

Parameters **timeout_ms** (*[type]*) – [description]

Returns [description]

Return type [type]

getTimeStamp()
[summary]

Returns [description]

Return type [type]

sendMessage(*msg*)
[summary]

Parameters **msg** (*[type]*) – [description]

mindaffectBCI.decoder.FileProxyHub.**testcase**(*filename*, *fs=200*, *fs_out=200*, *stopband=((45, 65), (0, 3), (25, -1))*, *order=4*)
[summary]

Parameters

- **filename** (*[type]*) – [description]
- **fs** (*int, optional*) – [description]. Defaults to 200.
- **fs_out** (*int, optional*) – [description]. Defaults to 200.
- **stopband** (*tuple, optional*) – [description]. Defaults to ((45,65),(0,3),(25,-1)).
- **order** (*int, optional*) – [description]. Defaults to 4.

mindaffectBCI.decoder.UtopiaDataInterface module

class mindaffectBCI.decoder.UtopiaDataInterface.**TransformerMixin**

Bases: object

fit(*X*)

transform(*X*)

class mindaffectBCI.decoder.UtopiaDataInterface.**UtopiaDataInterface**(*datawindow_ms=60000*, *msgwindow_ms=60000*, *data_preprocessor=None*, *stimulus_preprocessor=None*, *send_signalquality=True*, *timeout_ms=100*, *mintime_ms=50*, *fs=None*, *U=None*, *sample2timestamp='lower_bound_tr*
clientid=None)

Bases: object

Adaptor class for interfacing between the decoder logic and the data source

This class provides functionality to wrap a real time data and stimulus stream to make it easier to implement standard machine learning pipelines. In particular it provides streamed pre-processing for both EEG and stimulus streams, and ring-buffers for the same with time-stamp based indexing.

VERBOSITY = 1

add_sample_timestamps(*d: numpy.ndarray*, *timestamp: float*, *fs: float*)
add per-sample timestamp information to the data matrix

Parameters

- **d** (*np.ndarray*) – (t,d) the data matrix to attach time stamps to
- **timestamp** (*float*) – the timestamp of the last sample of d
- **fs** (*float*) – the nomional sample rate of d

Returns (t,d+1) data matrix with attached time-stamp channel

Return type np.ndarray

connect(*host=None*, *port=-1*, *queryifhostnotfound=True*)
[make a connection to the utopia host]

Parameters

- **host** (*[type], optional*) – [description]. Defaults to None.
- **port** (*int, optional*) – [description]. Defaults to -1.
- **queryifhostnotfound** (*bool, optional*) – [description]. Defaults to True.

Returns [description]

Return type [type]

extract_data_segment(*bgn_ts*, *end_ts=None*)
extract a segment of data based on a start and end time-stamp

Parameters

- **bgn_ts** (*float*) – segment start time-stamp
- **end_ts** (*float, optional*) – segment end time-stamp. Defaults to None.

Returns the data between these time-stamps, or None if timestamps invalid

Return type (np.ndarray)

extract_msgs_segment(*bgn_ts*, *end_ts=None*)
[extract the messages between start/end time stamps]

Parameters

- **bgn_ts** (*[type]*) – [description]
- **end_ts** (*[type], optional*) – [description]. Defaults to None.

Returns [description]

Return type [type]

extract_stimulus_segment(*bgn_ts*, *end_ts=None*)
extract a segment of the stimulus stream based on a start and end time-stamp

Parameters

- **bgn_ts** (*float*) – segment start time-stamp
- **end_ts** (*float, optional*) – segment end time-stamp. Defaults to None.

Returns the stimulus events between these time-stamps, or None if timestamps invalid

Return type (np.ndarray)

getNewMessages(*timeout_ms=0*)
[get new messages from the UtopiaHub]

Parameters **timeout_ms** (*int, optional*) – [description]. Defaults to 0.

Returns [description]

Return type [type]

getTimeStamp()
[summary]

Returns [description]

Return type [type]

initDataRingBuffer()
[initialize the data ring buffer, by getting some seed messages and datapackets to get the data sizes etc.]

Returns [description]

Return type [type]

initStimulusRingBuffer()
initialize the data ring buffer, by getting some seed messages and datapackets to get the data sizes etc.

isConnected()
[summary]

Returns [description]

Return type [type]

local_slope(*nsamp*, *data_ts*)
compute a robust local slope estimate

Parameters

- **nsamp** (*[type]*) – [description]

- **data_ts** (*[type]*) – [description]

Returns [description]

Return type [type]

plot_raw_preproc_data(*d_raw*, *d_preproc*, *ts*)
[debugging function to check the diff between the raw and pre-processed data]

Parameters

- **d_raw** (*[type]*) – [description]
- **d_preproc** (*[type]*) – [description]
- **ts** (*[type]*) – [description]

preprocess_message(*m: mindaffectBCI.utopiaclient.UtopiaMessage*)
[apply pre-processing to topia message before any more work]

Parameters **m** (UtopiaMessage) – [description]

Returns [description]

Return type [type]

processDataPacket(*m: mindaffectBCI.utopiaclient.DataPacket*)
[pre-process a datapacket message ready to be inserted into the ringbuffer]

Parameters **m** (DataPacket) – [description]

Returns [description]

Return type [type]

processStimulusEvent(*m: mindaffectBCI.utopiaclient.StimulusEvent*)
[pre-process a StimulusEvent message ready to be inserted into the stimulus ringbuffer]

Parameters **m** (StimulusEvent) – [description]

Returns [description]

Return type [type]

push_back_newmsgs(*oldmsgs*)
[put unprocessed messages back onto the newmessages queue]

Parameters **oldmsgs** (*[type]*) – [description]

run(*timeout_ms=30000*)
[test run the interface forever, just getting and storing data]

Parameters **timeout_ms** (*int, optional*) – [description]. Defaults to 30000.

sendMessage(*msg: mindaffectBCI.utopiaclient.UtopiaMessage*)
[send a UtopiaMessage to the utopia hub]

Parameters **msg** (UtopiaMessage) – [description]

update(*timeout_ms=None*, *mintime_ms=None*)
Update the tracking state w.r.t. the data source

By adding data to the data_ringbuffer, stimulus info to the stimulus_ringbuffer, and other messages to the messages ring buffer.

Args

timeout_ms [int] max block waiting for messages before returning

mintime_ms [int] min time to accumulate messages before returning

Returns

newmsgs [[newMsgs :UtopiaMessage]] list of the *new* utopia messages from the server

nsamp: int number of new data samples in this call Note: use data_ringbuffer[-nsamp:,...] to get the new data

nstimulus [int] number of new stimulus events in this call Note: use stimulus_ringbuffer[-nstimulus:,...] to get the new data

update_and_send_ElectrodeQualities(*d_raw: numpy.ndarray*, *d_preproc: numpy.ndarray*, *ts: int*)

[compute running estimate of electrode qality and stream it]

Parameters

- **d_raw** (*np.ndarray*) – [description]
- **d_preproc** (*np.ndarray*) – [description]
- **ts** (*int*) – [description]

update_electrode_powers(*d_raw: numpy.ndarray*, *d_preproc: numpy.ndarray*)

track exp-weighted-moving average centered power for 2 input streams – the raw and the preprocessed

Parameters

- **d_raw** (*np.ndarray*) – [description]
- **d_preproc** (*np.ndarray*) – [description]

Returns [description]

Return type [type]

class mindaffectBCI.decoder.UtopiaDataInterface.**butterfilt_and_downsample**(*stopband=((0, 5), (5, -1))*, *order: int = 6*, *fs: float = 250*, *fs_out: float = 60*, *ftype='butter'*)

Bases: *mindaffectBCI.decoder.UtopiaDataInterface.TransformerMixin*

Incremental streaming transformer to provide filtering and downsampling data transformations

Parameters **TransformerMixin** (*[type]*) – sklearn compatible transformer

fit(*X*, *fs: float = None*, *zi=None*)

[summary]

Parameters

- **X** (*[type]*) – [description]
- **fs** (*float, optional*) – [description]. Defaults to None.
- **zi** (*[type], optional*) – [description]. Defaults to None.

Returns [description]

Return type [type]

static testcase()
test the filt+downsample transformation filter by incremental calling

transform(*X*, *Y=None*)
[summary]

Parameters

- **X** (*[type]*) – [description]
- **Y** (*[type], optional*) – [description]. Defaults to None.

Returns [description]

Return type [type]

class mindaffectBCI.decoder.UtopiaDataInterface.**channel_power_standardizer**(*reg=0.0001*, *axis=-2*)

Bases: *mindaffectBCI.decoder.UtopiaDataInterface.TransformerMixin*

Incremental streaming tranformer to channel power normalization in an input stream

fit(*X*)
[summary]

Parameters **X** (*[type]*) – [description]

testcase(*dur=3*, *fs=100*, *blksize=10*)
[summary]

Parameters

- **dur** (*int, optional*) – [description]. Defaults to 3.
- **fs** (*int, optional*) – [description]. Defaults to 100.
- **blksize** (*int, optional*) – [description]. Defaults to 10.

transform(*X*)
add per-sample timestamp information to the data matrix

Parameters **X** (*float*) – the data to decorrelate

Returns the decorrelated data

Return type np.ndarray

class mindaffectBCI.decoder.UtopiaDataInterface.**power_tracker**(*halflife_mu_ms*, *halflife_power_ms*, *fs*, *car=True*)

Bases: *mindaffectBCI.decoder.UtopiaDataInterface.TransformerMixin*

Incremental streaming transformer from raw n-channel data, to exponientially smoothed channel powers

Parameters **TransformerMixin** (*[type]*) – sklearn compatiable transformer

fit(*X*)
[summary]

Parameters **X** ([type]) – [description]

Returns [description]

Return type [type]

hl2alpha(*hl*)
[summary]

Parameters **hl** ([type]) – [description]

Returns [description]

Return type [type]

mean()
[summary]

Returns [description]

Return type [type]

power()
[summary]

Returns [description]

Return type [type]

static testcase()
[summary]

transform(*X: numpy.ndarray*)
compute the exponientially weighted centered power of X

Parameters **X** (*np.ndarray*) – samples x channels data

Returns the updated per-channel power

Return type np.ndarray

class mindaffectBCI.decoder.UtopiaDataInterface.**stim2eventfilt**(*evtlabs=None*, *histlen=20*)
Bases: *mindaffectBCI.decoder.UtopiaDataInterface.TransformerMixin*

Incremental streaming transformer to transform a sequence of stimulus states to a brain event sequence

For example by transforming a sequence of stimulus intensities, to rising and falling edge events.

fit(*X*)
[summary]

Parameters **X** ([type]) – [description]

Returns [description]

Return type [type]

testcase()
test the stimulus transformation filter by incremental calling

transform(*X*)
[transform Stimulus-encoded to brain-encoded]

Parameters **X** ([type]) – [description]

Returns [description]

Return type [type]

class mindaffectBCI.decoder.UtopiaDataInterface.**temporal_decorrelator**(*order=10*, *reg=0.0001*, *eta=1e-05*, *axis=-2*)

Bases: *mindaffectBCI.decoder.UtopiaDataInterface.TransformerMixin*

Incremental streaming tranformer to decorrelate temporally channels in an input stream

fit(*X*)
[summary]

Parameters **X** (*[type]*) – [description]

testcase(*dur=3*, *fs=100*, *blksize=10*)
[summary]

Parameters

- **dur** (*int, optional*) – [description]. Defaults to 3.
- **fs** (*int, optional*) – [description]. Defaults to 100.
- **blksize** (*int, optional*) – [description]. Defaults to 10.

transform(*X*)
add per-sample timestamp information to the data matrix

Parameters

- **X** (*float*) – the data to decorrelate
- **nsamp** (*int*) – number of samples to interpolate

Returns the decorrelated data

Return type np.ndarray

mindaffectBCI.decoder.UtopiaDataInterface.**testERP**()
[summary]

mindaffectBCI.decoder.UtopiaDataInterface.**testElectrodeQualities**(*X*, *fs=200*, *pktsize=20*)
[summary]

Parameters

- **X** (*[type]*) – [description]
- **fs** (*int, optional*) – [description]. Defaults to 200.
- **pktsize** (*int, optional*) – [description]. Defaults to 20.

Returns [description]

Return type [type]

mindaffectBCI.decoder.UtopiaDataInterface.**testFileProxy**(*filename*, *fs_out=999*)
[summary]

Parameters

- **filename** (*[type]*) – [description]

- **fs_out** (*int, optional*) – [description]. Defaults to 999.

mindaffectBCI.decoder.UtopiaDataInterface.**testFileProxy2**(*filename*)
[summary]

Parameters **filename** (*[type]*) – [description]

mindaffectBCI.decoder.UtopiaDataInterface.**testPP**()
[summary]

mindaffectBCI.decoder.UtopiaDataInterface.**testRaw**()
[summary]

class mindaffectBCI.decoder.UtopiaDataInterface.**timestamp_interpolation**(*fs=None*, *sample2timestamp=None*, *max_delta=200*)

Bases: *mindaffectBCI.decoder.UtopiaDataInterface.TransformerMixin*

Incremental streaming tranformer to transform from per-packet time-stamps to per-sample timestamps with time-stamp smoothing, de-jittering, and dropped-sample detection.

fit(*ts*, *nsamp=1*)
[summary]

Parameters

- **ts** (*[type]*) – [description]
- **nsamp** (*int, optional*) – [description]. Defaults to 1.

testcase(*npkt=1000*, *fs=100*)
[summary]

Parameters

- **npkt** (*int, optional*) – [description]. Defaults to 1000.
- **fs** (*int, optional*) – [description]. Defaults to 100.

transform(*timestamp: float*, *nsamp: int = 1*)
add per-sample timestamp information to the data matrix

Parameters

- **timestamp** (*float*) – the timestamp of the last sample of d
- **nsamp** (*int*) – number of samples to interpolate

Returns (nsamp) the interpolated time-stamps

Return type np.ndarray

mindaffectBCI.decoder.analyse_datasets module

mindaffectBCI.decoder.analyse_datasets.**analyse_dataset**(*X: numpy.ndarray*, *Y: numpy.ndarray*, *coords*, *model: str = 'cca'*, *test_idx=None*, *cv=True*, *tau_ms: float = 300*, *fs: float = None*, *rank: int = 1*, *evtlabs=None*, *offset_ms=0*, *center=True*, *tuned_parameters=None*, *ranks=None*, *retrain_on_all=True*, ***kwargs*)

cross-validated training on a single datasets and decoing curve estimation

Parameters

- **X** (*np.ndarray*) – the X (EEG) sequence
- **Y** (*np.ndarray*) – the Y (stimulus) sequence
- **coords** (*[type]*) – array of dicts of meta-info describing the structure of X and Y
- **fs** (*float*) – the data sample rate (if coords is not given)
- **model** (*str, optional*) – The type of model to fit, as in *model_fitting.py*. Defaults to 'cca'.
- **cv** (*bool, optional*) – flag if we should train with cross-validation using the cv_fit method. Defaults to True.
- **test_idx** (*list-of-int, optional*) – indexs of test-set trials which are *not* passed to fit/cv_fit. Defaults to True.
- **tau_ms** (*float, optional*) – length of the stimulus-response in milliseconds. Defaults to 300.
- **rank** (*int, optional*) – rank of the decomposition in factored models such as cca. Defaults to 1.
- **evtlabs** (*[type], optional*) – The types of events to used to model the brain response, as used in *stim2event.py*. Defaults to None.
- **offset_ms** (*(2,)* – float, optional): Offset for analysis window from start/end of the event response. Defaults to 0.

Raises NotImplementedError – if you use for a model which isn't implemented

Returns the cv score for this dataset dc (tuple): the information about the decoding curve as returned by *decodingCurveSupervised.py* Fy (np.ndarray): the raw cv'd output-scores for this dataset as returned by *decodingCurveSupervised.py* clsfr (BaseSequence2Sequence): the trained classifier

Return type score (float)

mindaffectBCI.decoder.analyse_datasets.**analyse_datasets**(*dataset: str*, *model: str = 'cca'*, *dataset_args: dict = None*, *loader_args: dict = None*, *preprocess_args: dict = None*, *clsfr_args: dict = None*, *tuned_parameters: dict = None*, ***kwargs*)

analyse a set of datasets (multiple subject) and generate a summary decoding plot.

Parameters

- **dataset** (*[str]*) – the name of the dataset to load
- **model** (*str, optional*) – The type of model to fit. Defaults to 'cca'.
- **dataset_args** (*[dict], optional*) – additional arguments for get_dataset. Defaults to None.
- **loader_args** (*[dict], optional*) – additional arguments for the dataset loader. Defaults to None.
- **clsfr_args** (*[dict], optional*) – additional aguments for the model_fitter. Defaults to None.
- **tuned_parameters** (*[dict], optional*) – sets of hyper-parameters to tune by GridCVSearch

mindaffectBCI.decoder.analyse_datasets.**analyse_train_test**(*X: numpy.ndarray*, *Y: numpy.ndarray*, *coords*, *splits=1*, *label: str = ''*, *model: str = 'cca'*, *tau_ms: float = 300*, *fs: float = None*, *rank: int = 1*, *evtlabs=None*, *preprocess_args=None*, *clsfr_args: dict = None*, ***kwargs*)

analyse effect of different train/test splits on performance and generate a summary decoding plot.

Parameters

- **()** (*splits*) – list of list of train-test split pairs.
- **dataset** (*[str]*) – the name of the dataset to load
- **model** (*str, optional*) – The type of model to fit. Defaults to 'cca'.
- **dataset_args** (*[dict], optional*) – additional arguments for get_dataset. Defaults to None.
- **loader_args** (*[dict], optional*) – additional arguments for the dataset loader. Defaults to None.
- **clsfr_args** (*[dict], optional*) – additional aguments for the model_fitter. Defaults to None.
- **tuned_parameters** (*[dict], optional*) – sets of hyper-parameters to tune by GridCVSearch

mindaffectBCI.decoder.analyse_datasets.**debug_test_dataset**(*X*, *Y*, *coords=None*, *label=None*, *tau_ms=300*, *fs=None*, *offset_ms=0*, *evtlabs=None*, *rank=1*, *model='cca'*, *preprocess_args: dict = None*, *clsfr_args: dict = None*, *plotnormFy=False*, *triggerPlot=False*, ***kwargs*)

Debug a data set, by pre-processing, model-fitting and generating various visualizations

Parameters

- **X** (*nTrl,nSamp,d*) – The preprocessed EEG data
- **Y** (*nTrl,nSamp,nY*) – The stimulus information
- **coords** (*[type], optional*) – meta-info about the dimensions of X and Y. Defaults to None.
- **label** (*[type], optional*) – textual name for this dataset, used for titles and save-file names. Defaults to None.
- **tau_ms** (*int, optional*) – stimulus-response length in milliseconds. Defaults to 300.
- **fs** (*[type], optional*) – sample rate of X and Y. Defaults to None.
- **offset_ms** (*int, optional*) – offset for start of stimulus response w.r.t. stimulus time. Defaults to 0.
- **evtlabs** (*[type], optional*) – list of types of stimulus even to fit the model to. Defaults to None.
- **rank** (*int, optional*) – the rank of the model to fit. Defaults to 1.
- **model** (*str, optional*) – the type of model to fit. Defaults to 'cca'.
- **preprocess_args** (*dict, optional*) – additional arguments to send to the data pre-processor. Defaults to None.
- **clsfr_args** (*dict, optional*) – additional arguments to pass to the model fitting. Defaults to None.

Returns the cv score for this dataset dc (tuple): the information about the decoding curve as returned by *decodingCurveSupervised.py* Fy (np.ndarray): the raw cv'd output-scores for this dataset as returned by *decodingCurveSupervised.py* clsfr (BaseSequence2Sequence): the trained classifier

Return type score (float)

mindaffectBCI.decoder.analyse_datasets.**debug_test_single_dataset**(*dataset: str*, *filename: str = None*, *dataset_args=None*, *loader_args=None*, **args*, ***kwargs*)

run the debug_test_dataset for a single subject from dataset

Parameters

- **dataset** (*[str]*) – the dataset to load with get_dataset from *datasets.py*

- **filename** (*[str], optional*) – a specific filename regular expression to match to process. Defaults to None.

Returns the model fitted during the dataset testing

Return type clsfr [BaseSeq2seq]

mindaffectBCI.decoder.analyse_datasets.**flatten_decoding_curves**(*decoding_curves*)
take list of (potentially variable length) decoding curves and make into a single array

mindaffectBCI.decoder.analyse_datasets.**plot_trial_summary**(*X*, *Y*, *Fy*, *Fe=None*, *Py=None*, *fs=None*, *label=None*, *evtlabs=None*, *centerx=True*, *xspacing=10*, *sumFy=True*, *Yerr=None*)
generate a plot summarizing the inputs (X,Y) and outputs (Fe,Fe) for every trial in a dataset for debugging purposes

Parameters

- **X** (*nTrl,nSamp,d*) – The preprocessed EEG data
- **Y** (*nTrl,nSamp,nY*) – The stimulus information
- **Fy** (*nTrl,nSamp,nY*) – The output scores obtained by comping the stimulus-scores (Fe) with the stimulus information (Y)
- **Fe** (*(nTrl,nSamp,nY,nE), optional*) – The stimulus scores, for the different event types, obtained by combining X with the decoding model. Defaults to None.
- **Py** (*(nTrl,nSamp,nY), optional*) – The target probabilities for each output, derived from Fy. Defaults to None.
- **fs** (*float, optional*) – sample rate of X, Y, used to set the time-axis. Defaults to None.
- **label** (*str, optional*) – A textual label for this dataset, used for titles & save-files. Defaults to None.
- **centerx** (*bool, optional*) – Center (zero-mean over samples) X for plotting. Defaults to True.
- **xspacing** (*int, optional*) – Gap in X units between different channel lines. Defaults to 10.
- **sumFy** (*bool, optional*) – accumulate the output scores before plotting. Defaults to True.
- **Yerr** (*bool (nTrl,), optional*) – indicator for which trials the model made a correct prediction. Defaults to None.

mindaffectBCI.decoder.analyse_datasets.**run_analysis**()

mindaffectBCI.decoder.ccaViewer module

mindaffectBCI.decoder.ccaViewer.**ccaViewer**(**args*, ***kwargs*)

mindaffectBCI.decoder.ccaViewer.**parse_args**()

mindaffectBCI.decoder.ccaViewer.**run**(*ui: mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface*, *timeout_ms: float = inf*, *tau_ms: float = 500*, *offset_ms=(-15, 0)*, *evtlabs=None*, *ch_names=None*, *nstimulus_events: int = 600*, *rank: int = 3*, *reg=0.02*, *center: bool = True*, *host: str = '-'*, *stopband=None*, *out_fs=100*)

view the live CCA decomposition.

mindaffectBCI.decoder.dataset debugging module

mindaffectBCI.decoder.datasets module

mindaffectBCI.decoder.datasets.**brains_on_fire_online**()

mindaffectBCI.decoder.datasets.**cocktail**()

mindaffectBCI.decoder.datasets.**dataset_generator**(*dataset*, ***kwargs*)

generator for individual datasets from the given set of datasets

mindaffectBCI.decoder.datasets.**get_dataset**(*dsname*, **args*, ***kwargs*)

mindaffectBCI.decoder.datasets.**lowlands**()

generate the directory+filename info for the lowlands noisetagging dataset

mindaffectBCI.decoder.datasets.**mTRF_audio**()

mindaffectBCI.decoder.datasets.**mark_EMG**()

mindaffectBCI.decoder.datasets.**mindaffectBCI**(*exptdir*, ***args*)

mindaffectBCI.decoder.datasets.**ninapro_db2**()

mindaffectBCI.decoder.datasets.**openBMI**(*dstype='SSVEP'*)

mindaffectBCI.decoder.datasets.**p300_prn**(*label: str = None*)

generate dataset+filename for p300-prn2

mindaffectBCI.decoder.datasets.**plos_one**()

generate the directory+filename info for the plos_one noisetagging dataset

mindaffectBCI.decoder.datasets.**tactileP3**()

generate dataset+filename for tactile P3

mindaffectBCI.decoder.datasets.**tactile_PatientStudy**()

generate dataset+filename for p300-prn2

mindaffectBCI.decoder.datasets.**test_loader**(*loadfn*, *filenames*, *dataroot*, ***kwargs*)

mindaffectBCI.decoder.datasets.**testcase**()

mindaffectBCI.decoder.datasets.**testdataset**(*fn*, ***args*)

a conforming toy dataset loader

mindaffectBCI.decoder.datasets.**toy**()

make a toy dataset for testing

mindaffectBCI.decoder.datasets.**twofinger**()

mindaffectBCI.decoder.dec_test module

mindaffectBCI.decoder.decoder module

mindaffectBCI.decoder.decoder.**combine_Ptgt**(*pvals_objIDs*)
combine target probabilities in a correct way

Parameters **pvals_objIDs** (*list (pval,objId)*) – list of Ptgt,objID pairs for outputs at different time points.

Returns target probabilities (np.ndarray (outputs,)) : object IDs for the targets

Return type (np.ndarray (outputs,))

mindaffectBCI.decoder.decoder.**dataset_to_XY_ndarrays**(*dataset*)
convert a dataset, consisting of a list of pairs of time-stamped data and stimulus events, to 3-d matrices of X=(trials,samples,channels) and Y=(trials,samples,outputs)

Parameters **dataset** (*[type]*) – list of pairs of time-stamped data and stimulus events

Returns the per-trial data Y (tr,samp,nY): the per-trial stimulus, with sample rate matched to X X_ts (tr,samp): the time-stamps for the smaples in X Y_ts (tr,samp): the time-stamps for the stimuli in Y

Return type X (tr,samp,d)

mindaffectBCI.decoder.decoder.**doCalibrationSupervised**(*ui: mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterfac clsfr: mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence*, ***kwargs*)
do a calibration phase = basically just extract the training data and train a classifier from the utopiaInterface

Parameters

- **ui** (UtopiaDataInterface) – buffered interface to the data and stimulus streams
- **clsfr** (BaseSequence2Sequence) – the classifier to use to fit a model to the calibration data
- **cv** (*int, optional*) – the number of cross-validation folds to use for model generalization performance estimation. Defaults to 2.
- **prior_dataset** (*[type], optional*) – data-set from a previous calibration run, used to accumulate data over subsequent calibrations. Defaults to None.
- **ranks** (*tuple, optional*) – a list of model ranks to optimize as hyperparameters. Defaults to (1,2,3,5).

Returns the gathered calibration data X : the calibration data as a 3-d array (tr,samp,d) Y : the calibration stimulus as a 3-d array (tr,samp,num-outputs)

Return type dataset [type]

mindaffectBCI.decoder.decoder.**doModelFitting**(*clsfr: mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence*, *dataset*, *cv: int = 2*, *prior_dataset=None*, *ranks=(1, 2, 3, 5)*, *fs: float = None*, *n_ch: int = None*, ***kwargs*)
fit a model given a dataset

Parameters

- **clsfr** (BaseSequence2Sequence) – the classifier to use to fit a model to the calibration data
- **cv** (*int, optional*) – the number of cross-validation folds to use for model generalization performance estimation. Defaults to 2.
- **prior_dataset** (*[type], optional*) – data-set from a previous calibration run, used to accumulate data over subsequent calibrations. Defaults to None.
- **ranks** (*tuple, optional*) – a list of model ranks to optimize as hyperparameters. Defaults to (1,2,3,5).

Returns the estimated model generalization performance on the training data. dataset [type]: the gathered calibration data X : the calibration data as a 3-d array (tr,samp,d) Y : the calibration stimulus as a 3-d array (tr,samp,num-outputs)

Return type perr (float)

mindaffectBCI.decoder.decoder.**doPrediction**(*clsfr: mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence*, *data*, *stimulus*, *prev_stimulus=None*)

given the current trials data, apply the classifier and decoder to make target predictions

Parameters

- **clsfr** (BaseSequence2Sequence) – the trained classifier to apply to the data
- **data** (*np.ndarray (time,channels)*) – the pre-processed EEG data
- **stimulus** (*np.ndarray (time,outputs)*) – the raw stimulus information
- **prev_stimulus** (*np.ndarray, optional*) – previous stimulus before stimulus – poss needed for correct event coding. Defaults to None.

Returns Fy scores for each output at each time-point

Return type (np.ndarray (time,outputs))

mindaffectBCI.decoder.decoder.**doPredictionStatic**(*ui: mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface*, *clsfr: mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence*, *model_apply_type: str = 'trial'*, *timeout_ms: float = None*, *block_step_ms: float = 100*, *maxDecisLen_ms: float = 8000*)

do the prediction stage = basically extract data/msgs from trial start and generate a prediction from them '"

Parameters

- **ui** (UtopiaDataInterface) – buffered interface to the data and stimulus streams
- **clsfr** (BaseSequence2Sequence) – the trained classification model
- **maxDecisLen_ms** (*float, optional*) – the maximum amount of data to use to make a prediction, i.e. prediction sliding window size. Defaults to 8000

mindaffectBCI.decoder.decoder.**getCalibration_dataset**(*ui: mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface*

extract a labelled dataset from the utopiaInterface, which are trials between modechange messages

Parameters **ui** (UtopiaDataInterface) – the data interface object

Returns list of pairs of time-stamped data and stimulus information as 2d (time,ch) (or (time,output)) numpy arrays

Return type (list (data,stimulus))

mindaffectBCI.decoder.decoder.**get_trial_start_end**(*msgs*, *start_ts=None*)
get the start+end times of the trials in a utopia message stream

Parameters

- **msgs** (*[mindaffectBCI.UtopiaMessage]*) – list of messages recenty recieved
- **start_ts** (*float, optional*) – time-stamp for start of *current* trial. Defaults to None.

Returns list of completed trial (start_ts,end_ts) time-stamp tuples (float): start_ts for trial started but not finished (list UtopiaMessage): list of unprocessed messages

Return type (list (start_ts,end_ts))

mindaffectBCI.decoder.decoder.**load_previous_dataset**(*f: str*)
search standard directory locations and load a previously saved (pickled) calibration dataset

Parameters **f** (*str, file-like*) – buffered interface to the data and stimulus streams

Returns list of stimulus,data pairs for each trial

Return type (list of (data,stimulus))

mindaffectBCI.decoder.decoder.**parse_args**()

mindaffectBCI.decoder.decoder.**plot_trial_summary**(*Ptgt*, *Fy=None*, *Py=None*, *fs: float = None*)
Plot a summary of the trial decoding information

Parameters

- **Ptgt** (*np.ndarray*) – the current output probabilities
- **Fy** (*np.ndarray*) – the raw output scores over time
- **Py** (*np.ndarray*) – the raw probabilities for each target over time
- **fs** (*float, optional*) – the data sample rate. Defaults to None.

mindaffectBCI.decoder.decoder.**redraw_plots**()

mindaffectBCI.decoder.decoder.**run**(*ui: mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface = None*, *clsfr: mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence = None*, *msg_timeout_ms: float = 100*, *host: str = None*, *prior_dataset: str = None*, *tau_ms: float = 450*, *offset_ms: float = 0*, *out_fs: float = 100*, *evtlabs=None*, *stopband=((45, 65), (5.5, 25, 'bandpass'))*, *ftype='butter'*, *order: int = 6*, *cv: int = 5*, *prediction_offsets=None*, *logdir=None*, *calplots: bool = False*, *predplots: bool = False*, *label: str = None*, ***kwargs*)
run the main decoder processing loop

Parameters

- **ui** (UtopiaDataInterface, *optional*) – The utopia data interface class. Defaults to None.
- **clsfr** (BaseSequence2Sequence, *optional*) – the classifer to use when model fitting. Defaults to None.
- **msg_timeout_ms** (*float, optional*) – timeout for getting new messages from the data-interface. Defaults to 100.

- **host** (*str, optional*) – hostname for the utopia hub. Defaults to None.
- **tau_ms** (*float, optional*) – length of the stimulus response. Defaults to 400.
- **offset_ms** (*float, optiona*) – offset in ms to shift the analysis window. Use to compensate for response lag. Defaults to 0.
- **stopband** (*tuple, optional*) – temporal filter specification for *UtopiaDataInterface.butterfilt_and_downsample*. Defaults to ((45,65),(5.5,25,'bandpass'))
- **ftype** (*str, optional*) – type of temporal filter to use. Defaults to 'butter'.
- **logdir** (*str, optional*) – location to save output files. Defaults to None.
- **order** (*int, optional*) – order of temporal filter to use. Defaults to 6.
- **out_fs** (*float, optional*) – sample rate after the pre-processor. Defaults to 100.
- **evtlabs** (*tuple, optional*) – the brain event coding to use. Defaults to None.
- **calplots** (*bool, optional*) – flag if we make plots after calibration. Defaults to False.
- **predplots** (*bool, optional*) – flag if we make plots after each prediction trial. Defaults to False.
- **prior_dataset** (*[str,(dataset)]*) – calibration data from a previous run of the system. Used to pre-seed the model. Defaults to None.
- **prediction_offsets** (*[ListInt], optional*) – a list of stimulus offsets to try at prediction time to cope with stimulus timing jitter. Defaults to None.

mindaffectBCI.decoder.decoder.**send_prediction**(*ui: mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface*, *Ptgt*, *used_idx=None*, *timestamp: int = -1*)

Send the current prediction information to the utopia-hub

Parameters

- **ui** (UtopiaDataInterface) – the interface to the data-hub
- **Ptgt** (*np.ndarray (outputs,)*) – the current distribution of target probabilities over outputs
- **used_idx** (*np.ndarray, optional*) – a set of output indices currently used. Defaults to None.
- **timestamp** (*int, optional*) – time stamp for which this prediction applies. Defaults to -1.

mindaffectBCI.decoder.decoder.**strip_unused**(*Y*)

strip unused outputs from the stimulus info in Y

Parameters **Y** (*np.ndarray (time,outputs)*) – the full stimulus information, potentionally with many unused outputs

Returns Y with unused outputs removed

Return type (np.ndarray (time,used-outputs))

mindaffectBCI.decoder.decodingCurveSupervised module

mindaffectBCI.decoder.decodingCurveSupervised.**compute_decoding_curve**(*Fy: numpy.ndarray*, *objIDs*, *integerationLengths*, ***kwargs*)

compute the decoding curves from the given epoch+output scores in Fy

Parameters

- **(float** (*objIDs*) – ((nM,)nTrl,nEp,nY)) : per-epoch output scores
- **(float** – (nY,)) : the objectIDs for the outputs in Fy
- **integerationLengths** (*float (nInt,)*) – a list of integeration lengths to compute peformance at

Returns bool) : flag if prediction was *incorrect* at this integeration length for this trial Perr (nTrl,nInt : float) : compute probability of error for this integeration length and trial aveProbErr (nInt: float) : average error probablility at this integeration length aveProbErrEst (nInt: float): average estimated error probability for this integeration length

Return type Yerr (nTrl,nInt

mindaffectBCI.decoder.decodingCurveSupervised.**compute_stopping_curve**(*nInt*, *integerationLengths*, *Perr*, *Yerr*)

compute the stopping curve – which is the performance at times when stopping threshold (Perr) is passed

Parameters

- **nInt** (*int*) – number of time points to compute the stopping curve at
- **integerationLengths** (*list int*) – the set of integeration lengths at which stopping curve is computed
- **Perr** (*nTrl,nInt*) – Probability of error at each time point
- **Yerr** (*nTrl,nInt*) – For each time point if the 'best' prediction is correct or not

Returns [description]

Return type [type]

mindaffectBCI.decoder.decodingCurveSupervised.**decodingCurveSupervised**(*Fy*, *objIDs=None*, *nInt=(30, 25)*, *dedup0=True*, ***kwargs*)

Compute a decoding curve, i.e. mistake-probability over time for probability based stopping from the per-epoch output scores

Parameters

- **Fy** (*nModel,nTrl,nEp,nY*) – similarity score for each input epoch for each output N.B. Supervised decoder only has 1 model!!!
- **objIDs** (*nY,*) – mapping from rows of Fy to output object IDs. N.B. assumed objID==0 is true target N.B. if objIDs > size(Fy,2), then additional virtual outputs are added
- **nInt** (*2,*) – the number of integeration lengths to use, numThresholds. Defaults to ([30,25])

Returns the actual integeration lengths in samples ProbErr (float (nInt,)) : empherical error probablility at this integeration length ProbErrEst (float (nInt,)) : decoder estimate of the error rate for each integeration length StopPerr (nInt,) : error rate at this average trial length when using ProbErrEst-thresholding based stopping StopThresh (nInt,) : ProbErrEst threshold used to get this average trial length. Yerr (bool (nTrl,nInt)) : flag if prediction was *incorrect* at this integeration length for this trial Perr (float (nTrl,nInt)) : compute probability of incorrect prediction for this integeration length and trial

Return type integerationLengths (int (nInt,))

mindaffectBCI.decoder.decodingCurveSupervised.**plot_decoding_curve**(*integerationLengths*, *aveProbErr*, **args*)

plot the decoding curve

Parameters

- **integerationLengths** (*[type]*) – [description]
- **aveProbErr** (*[type]*) – [description]

mindaffectBCI.decoder.decodingCurveSupervised.**print_decoding_curve**(*integerationLengths*, *aveProbErr*, *aveProbErrEst=None*, *stopYerr=None*, *stopPerrThresh=None*)

[summary]

Parameters

- **integerationLengths** (*[type]*) – [description]
- **aveProbErr** (*[type]*) – [description]
- **aveProbErrEst** (*[type], optional*) – [description]. Defaults to None.
- **stopYerr** (*[type], optional*) – [description]. Defaults to None.
- **stopPerrThresh** (*[type], optional*) – [description]. Defaults to None.

Returns [description]

Return type [type]

mindaffectBCI.decoder.decodingCurveSupervised.**testcase**()

[summary]

mindaffectBCI.decoder.decodingSupervised module

mindaffectBCI.decoder.decodingSupervised.**decodingSupervised**(*Fy*, *softmaxscale=3.5*, *marginalizemodels=True*, *marginalizedecis=False*, *prior=None*, *nocontrolamplitude=None*, *tiebreaking_noise=0.001*, ***kwargs*)

true-target estimator and error-probility estimator for each trial

Parameters

- **Fy** (*nModel,nTrl,nEp,nY*) – The output scores.
- **softmaxscale** (*float, optional*) – the scale length to pass to zscore2Ptgt_softmax.py, this is the inverse *noise* sigma. Defaults to 3.5.
- **badFyThresh** (*int, optional*) – threshold for detction of bad Fy entry, in std-dev. Defaults to 4.
- **centFy** (*bool, optional*) – bool, do we center Fy before computing *important* (true). Defaults to True.
- **detrendFy** (*bool, optional*) – [description]. Defaults to True.
- **nEpochCorrection** (*int, optional*) – int, number of epochs to use a base for correction of number epochs in the sum. such that: zscore = Fy / (sigma * (1+1/(nEpoch/nEpochErrorCorrection))), basically, nEpoch < nEpochCorrection have highly increased Perr, so unlikely to be selected. Defaults to 100.
- **minDecisLen** (*int, optional*) –

  int, number of epochs to use as base for distribution of time-based decision points.
  i.e. decisions at, [1,2,4,...2^n]*exptDistDecis

  OR: minDecisLen<0 => decision point every abs(minDeicsLen) epochs. Defaults to 0.
- **maxDecisLen** (*int, optional*) – maximum number of epochs for a decision. Defaults to 0.
- **bwdAccumulate** (*bool, optional*) – accumulate data backwards from last epoch gathered. Defaults to False.
- **marginalizemodels** (*bool, optional*) – [description]. Defaults to True.
- **marginalizedecis** (*bool, optional*) – [description]. Defaults to False.
- **prior** (*[type], optional*) – [description]. Defaults to None.
- **nocontrolamplitude** (*[type], optional*) – [description]. Defaults to None.
- **priorsigma** (*tuple, optional*) – (sigma,N) prior estimate of sigma2 and number pseudo-points. Defaults to (-1,120).
- **tiebreaking_noise** (*[type], optional*) – [description]. Defaults to 1e-3.

Raises NotImplementedError – [description]

Returns the most likely / minimum error output for each decision point Perr (nTrl,nDecis): the probability that this selection is an ERROR for each decision point Ptgt (nTrl,nDecis,nY): the probability each target is the true target for each decision point decisMdl, decisEp

Return type

Yest (nTrl,nDecis)

Copyright (c) MindAffect B.V. 2018

mindaffectBCI.decoder.decodingSupervised.**decodingSupervised_streamed**(*Fy*, *softmaxscale=3*, *nocontrolamplitude=None*, *marginalizemodels=False*)

true-target estimator and error-probility estimator for each trial Inputs:

Fy - (nModel,nTrl,nEp,nY) [#Y x #Epoch x #Trials x nModels] softmaxscale - the scale length to pass to zscore2Ptgt_softmax.py badFyThresh - threshold for detction of bad Fy entry, in std-dev centFy - bool, do we center Fy before computing *important* (true) nEpochCorrection - int, number of epochs to use a base for correction of number epochs in the sum.

such that: zscore = Fy / (sigma * (1+1/(nEpoch/nEpochErrorCorrection))), basically, nEpoch < nEpochCorrection have highly increased Perr, so unlikely to be selected

minDecisLen - int, number of epochs to use as base for distribution of time-based decision points.

i.e. decisions at, [1,2,4,...2^n]*exptDistDecis

OR: minDecisLen<0 => decision point every abs(minDeicsLen) epochs

bwdAccumulate - [bool], accumulate data backwards from last epoch gathered maxDecisLen - maximum number of epochs for a decision

Outputs: Yest - (nTrl,nDecis) [nDecis x nTrl] the most likely / minimum error output for each decision point Perr - (nTrl,nDecis) [nDecix x nTrl] the probability that this selection is an ERROR for each decision point Ptgt - (nTrl,nDecis,nY) [nY x nDecis x nTrl] the probability each target is the true target for each decision point

Copyright (c) MindAffect B.V. 2018

mindaffectBCI.decoder.decodingSupervised.**testcase**()

mindaffectBCI.decoder.devent2stimsequence module

mindaffectBCI.decoder.devent2stimsequence.**devent2stimSequence**(*devents*)

convert a set of STIMULUSEVENT messages into a stimulus-sequence array with stimulus-times as expected by the utopia RECOGNISER

Inputs:

devents - [nEvt]:UtopiaMessage list of UtopiaMessage messages

i.e. a decoded utopia STIMULUSEVENT message, should be of type { msgID:'E'byte, timeStamp:int, objIDs:(nObj:byte), objState:(nObj:byte) }

Outputs: Me - (nEvt,nY :int) The extract stimulus sequence objIDs - (nY :byte) The set of used object IDs stimTimes_ms - (nEvt :int) The timestamp of each event in milliseconds isistimEvent - (nEp :bool) Indicator which input events are stimulus events

Copyright (c) MindAffect B.V. 2018

`mindaffectBCI.decoder.devent2stimsequence.`**`upsample_stimseq`**(*sample_ts*, *ss*, *stimulus_ts*, *objIDs=None*, *usedobjIDs=None*, *trlen=None*)

upsample a set of timestamped stimulus states to a sample rate WARNING: assumes sample_ts and stimulus_ts are in *ascending* sorted order!

mindaffectBCI.decoder.erpViewer module

`mindaffectBCI.decoder.erpViewer.`**`erpViewer`**(*ui: mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface*, *timeout_ms: float = inf*, *tau_ms: float = 500*, *offset_ms=(-15, 0)*, *evtlabs=None*, *ch_names=None*, *nstimulus_events: int = 600*, *rank=3*, *center=True*)

simple sig-viewer using the ring-buffer for testing

mindaffectBCI.decoder.lower_bound_tracker module

`class` `mindaffectBCI.decoder.lower_bound_tracker.`**`lower_bound_tracker`**(*window_size=200*, *C=(0.1, None)*, *outlier_thresh=(1, None)*, *step_size=10*, *step_threshold=2*, *a0=1*, *b0=0*, *warmup_size=10*)

Bases: `object`

sliding window lower bound trend tracker, which fits a line to the lower-bound of the inputs x,y

`append`(*X*, *Y*)

`fit`(*X*, *Y*)

`getX`(*y*)

`getY`(*x*)

`reset`(*keep_model=False*)

reset the data for model fitting

Parameters **`keep_model`** (*`bool, optional`*) – keep the model as well as data. Defaults to False.

static testcase(*savefile='-'*)

transform(*X*, *Y*)

update()
update the model based on the data in the sliding window

mindaffectBCI.decoder.model_fitting module

class mindaffectBCI.decoder.model_fitting.**BaseEstimator**
Bases: object

class mindaffectBCI.decoder.model_fitting.**BaseSequence2Sequence**(*evtlabs=('re', 'fe')*, *tau=18*, *offset=0*, *priorweight=200*, *startup_correction=100*, *prediction_offsets=None*, *minDecisLen=0*, *bwdAccumulate=False*, *verb=0*)
Bases: *mindaffectBCI.decoder.model_fitting.BaseEstimator*, *mindaffectBCI.decoder.model_fitting.ClassifierMixin*

Base class for sequence-to-sequence learning. Provides, prediction and scoring functions, but not the fitting method

static audc_score(*Fy*, *marginalizemodels=True*)
compute area under decoding curve score from Fy, *assuming* Fy[:,:,0] is the *true* classes score

clear()

cv_fit(*X*, *Y*, *cv=5*, *fit_params: dict = {}*, *verbose: bool = 0*, *return_estimator: bool = True*, *calibrate_softmax: bool = True*, *retrain_on_all: bool = True*)
Cross-validated fit the model for generalization performance estimation

N.B. write our own as sklearn doesn't work for getting the estimator values for structured output.

Parameters

- **X** (*[type]*) – [description]
- **Y** (*[type]*) – [description]
- **cv** (*int, optional*) – the number of folds to use, or a fold generator. Defaults to 5.
- **fit_params** (*dict, optional*) – additional parameters to pass to the self.fit(X,Y,...) function. Defaults to dict().
- **verbose** (*bool, optional*) – flag for model fitting verbosity. Defaults to 0.
- **return_estimator** (*bool, optional*) – should we return the cross-validated predictions. Defaults to True.
- **calibrate_softmax** (*bool, optional*) – after fitting the model, should we use the cross-validated predictions to calibrate the probability estimates. Defaults to True.
- **dedup0** (*bool, optional*) – should we de-duplicate copies of the 'true-target'. Defaults to True.

- **retrain_on_all** (*bool, optional*) – should we retrain the model on all the data after the cv folds. Defaults to True.

Returns dictionary with the results

Return type results (dict)

decode_proba(*Fy*, *minDecisLen=0*, *bwdAccumulate=True*, *marginalizemodels=True*, *marginalizedecis=False*)

Convert stimulus scores to stimulus probabities of being the target

Parameters

- **Fy** (*np.ndarray (tr,samp,nY)*) – the multi-trial stimulus sequence scores
- **minDecisLen** (*int,optional*) – minimum number of samples on which to make a prediction. Defaults to 0.
- **bwdAccumulate** (*bool, optional*) – accumulate information backwards in the trials. Defaults to True.
- **marginalizemodels** (*bool,optional*) – flag if we should marginalize over models when have multiple prediction models. Defaults to True.
- **marginalizedecis** (*bool, optional*) – flag if we should marginalize over decision points when have multiple. Defaults to False.

Raises *NotFittedError* – [description]

Returns score for each decision point in each trial for each output. Higher score means more 'likely' to be the 'true' target

Return type Ptgt (np.ndarray (tr,nDecis,nY))

fit(*X*, *Y*)

fit model mapping 2 multi-dim time series: X = (tr, samp, d), Y = (tr, samp, e)

is_fitted()

plot_model(***kwargs*)

predict(*X*, *Y*, *dedup0=True*, *prevY=None*, *offsets=None*)

Generate predictions with the fitted model for the paired data + stimulus-sequences

N.B. this implementation assumes linear coefficients in **W_** (nM,nfilt,d) and **R_** (nM,nfilt,nE,tau)

Parameters

- **X** (*np.ndarray (tr,samp,d)*) – the multi-trial data
- **Y** (*np.ndarray (tr,samp,nY)*) – the multi-trial stimulus sequences
- **dedup0** (*[type], optional*) – remove duplicates of the Yidx==0, i.e. 1st, assumed true, output of Y. Defaults to True.
- **prevY** (*[type], optional*) – previous stimulus sequence information. for partial incremental calls. Defaults to None.
- **offsets** (*[ListInt], optional*) – list of offsets in Y to try when decoding, to override the class variable. Defaults to None.

Raises *NotFittedError* – raised if try to predict without first fitting

Returns score for each output in each trial. Higher score means more 'likely' to be the 'true' target

Return type Fy (np.ndarray (mdl,tr,samp,nY)

predict_proba(*X*, *Y*, *marginalizemodels=True*, *marginalizedecis=True*, *startup_correction=100*, *minDecisLen=None*, *bwdAccumulate=True*, *dedup0=True*, *prevY=None*)

Predict the probability of each output for paired data/stimulus sequences

Parameters

- **X** (*np.ndarray (tr,samp,d)*) – the multi-trial data
- **Y** (*np.ndarray (tr,samp,nY)*) – the multi-trial stimulus sequences
- **dedup0** (*bool, optional*) – remove duplicates of the Yidx==0, i.e. 1st, assumed true, output of Y. Defaults to True.
- **prevY** (*np.ndarray, optional*) – previous stimulus sequence information. for partial incremental calls. Defaults to None.
- **minDecisLen** (*int,optional*) – minimum number of samples on which to make a prediction
- **marginalizemodels** (*bool,optional*) – flag if we should marginalize over models when have multiple prediction models. Defaults to True.
- **marginalizedecis** (*bool, optional*) – flag if we should marginalize over decision points when have multiple. Defaults to False.

Raises *NotFittedError* – [description]

Returns Probability of each output being the target. Higher score means more 'likely' to be the 'true' target

Return type Ptgt (np.ndarray (tr,nDecis,nY)

score(*X*, *Y*)

score this model on this data, N.B. for model_selection higher is *better*

stim2event(*Y*, *prevY=None*)

transform Stimulus-encoded to brain-encoded, if needed

transform(*X*)

transform raw data into raw stimulus scores by convolving X with the model, i.e. Fe = X (*) (W*R)

Parameters **X** (*np.ndarray (nTrl,nSamp,d)*) – The raw eeg data.

Returns The raw stimulus scores.

Return type Fe (np.ndarray (nTrl,nSamp,nE))

class mindaffectBCI.decoder.model_fitting.**BwdLinearRegression**(*evtlabs=('re', 're')*, *tau=18*, *offset=0*, *reg=None*, *rcond=1e-05*, *badEpThresh=6*, *center=True*, ***kwargs*)

Bases: *mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence*

Sequence 2 Sequence learning using backward linear regression W*X = Y

fit(*X*, *Y*)

fit 2 multi-dim time series: X = (tr, samp, d), Y = (tr, samp, e)

class mindaffectBCI.decoder.model_fitting.**ClassifierMixin**

Bases: object

class mindaffectBCI.decoder.model_fitting.**FwdLinearRegression**(*evtlabs=('re', 're')*, *tau=18*, *offset=0*, *reg=None*, *rcond=1e-06*, *badEpThresh=6*, *center=True*, ***kwargs*)

Bases: *mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence*

Sequence 2 Sequence learning using forward linear regression X = A*Y

fit(*X*, *Y*)

fit 2 multi-dim time series: X = (tr, samp, d), Y = (tr, samp, e)

class mindaffectBCI.decoder.model_fitting.**LinearSklearn**(*clsfr*, *labelizeY=False*, *ignore_unlabelled=True*, *badEpThresh=None*, ***kwargs*)

Bases: *mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence*

Wrap a normal sk-learn classifier for sequence to sequence learning

fit(*X*, *Y*)

fit model mapping 2 multi-dim time series: X = (tr, samp, d), Y = (tr, samp, e)

static sklearn_fit(*X*, *Y*, *clsfr*, *tau*, *offset*, *labelizeY=False*, *ignore_unlabelled=True*, *badEpThresh=None*, *verb=0*)

fit 2 multi-dim time series: X = (tr, samp, d), Y = (tr, samp, e) using a given sklearn linear_model method

class mindaffectBCI.decoder.model_fitting.**MultiCCA**(*evtlabs=('re', 're')*, *tau=18*, *offset=0*, *rank=1*, *reg=(1e-08, None)*, *rcond=(0.0001, 1e-08)*, *badEpThresh=6*, *symmetric=False*, *center=True*, *CCA=True*, *priorweight=200*, *startup_correction=100*, *prediction_offsets=None*, *minDecisLen=100*, *bwdAccumulate=False*, ***kwargs*)

Bases: *mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence*

Sequence 2 Sequence learning using CCA as a bi-directional forward/backward learning method

cv_fit(*X*, *Y*, *cv=5*, *fit_params: dict = {}*, *verbose: bool = 0*, *return_estimator: bool = True*, *calibrate_softmax: bool = True*, *retrain_on_all: bool = True*, *ranks=None*)

cross validated fit to the data. N.B. write our own as sklearn doesn't work for getting the estimator values for structured output.

fit(*X*, *Y*, *stimTimes=None*)

fit 2 multi-dim time series: X = (tr, samp, d), Y = (tr, samp, Y)

fit_b(*X*)

fit the bias parameter given the other parameters and a dataset

Parameters **X** (*np.ndarray*) – the target data

exception mindaffectBCI.decoder.model_fitting.**NotFittedError**

Bases: Exception

class mindaffectBCI.decoder.model_fitting.**StratifiedKFold**(*n_splits*)

Bases: object

split(*X*, *Y*)

mindaffectBCI.decoder.model_fitting.**plot_model_weights**(*model: mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence*, *ch_names=None*)

mindaffectBCI.decoder.model_fitting.**testLeadLag**()

mindaffectBCI.decoder.model_fitting.**testcase**(*dataset='toy'*, *loader_args={}*)

mindaffectBCI.decoder.model_fitting.**visualize_Fy_Py**(*Fy*, *Py*)

mindaffectBCI.decoder.multipleCCA module

mindaffectBCI.decoder.multipleCCA.**cvSupervised**(*Xe*, *Me*, *stimTimes*, *evtlabs=('re', 'fe')*, *n_splits=10*, *rank=1*)

do a cross-validated training of a multicca model using sklearn

mindaffectBCI.decoder.multipleCCA.**filterbank_testcase**()

mindaffectBCI.decoder.multipleCCA.**fit_predict**(*X*, *Y*, *tau=10*, *uss_args={}*, *mcca_args={}*)

mindaffectBCI.decoder.multipleCCA.**multipleCCA**(*Cxx=None*, *Cxy=None*, *Cyy=None*, *reg=1e-08*, *rank=1*, *CCA=True*, *rcond=0.0001*, *symetric=False*)

Compute multiple CCA decompositions using the given summary statistics

[J,W,R]=multiCCA(Cxx,Cxy,Cyy,regx,regy,rank,CCA)

Inputs: Cxx = (d,d) current data covariance Cxy = (nM,nE,tau,d) current per output ERPs Cyy = (nM,nE,tau,nE,tau) current response covariance for each output reg = (1,) :float or (2,):float linear weighting reg strength or separate values for Cxx and Cyy

OR (d,) regularisation ridge coefficients for Cxx (0)

rank= [1x1] number of top cca components to return (1) CCA = [bool] or (2,):bool flag if we normalize the spatial dimension. (true) rcond = [float] tolerance for singular eigenvalues. (1e-4)

or [2x1] separate rcond for Cxx (rcond(1)) and Cyy (rcond(2)) or [2x1] (-1<-0) negative values = keep this fraction of eigen-values or (2,1) <-1 keep this many eigenvalues

symetric = [bool] us symetric whitener?

Outputs: J = (nM,) optimisation objective scores W = (nM,rank,d) spatial filters for each output R = (nM,rank,nE,tau) responses for each stimulus event for each output

Examples

Supervised CCA Y = (nEp/nSamp,nY,nE) indicator for each event-type and output of it's type in each epoch X = (nEp/nSamp,tau,d) sliced pre-processed raw data into per-stimulus event responses stimTimes = (nEp) # sample number for each epoch or None if sample rate Cxx,Cxy,Cyy=updateSummaryStatistics(X, Y, stimTimes) J,w, r=multiCCA(Cxx, Cxy, Cyy, reg) # w=(nM,d)[dx1],r=(nM,nE,tau)[tau x nE] are the found spatial,spectral filters # incremental zeroTrain Cxx=[];Cxy=[];Cyy=[]; prediction=[]; while isCalibration:

Slicing newX,newE,stimTimes= preprocessnSliceTrial(); #function which get 1 trials data+stim # Model Fitting Cxx, Cxy, Cyy= updateSummaryStatistics(newX, newE, stimTimes, Cxx, Cxy, Cyy); J, w, r = multipleCCA(Cxx, Cxy, Cyy, regx, regy) # w=[dx1],r=[tau x nE] are the found spatial,spectral filters

end # re-compute with current state, can try different regulisors, e.g. for adaptive bad-ch regx = updateRegFromBadCh(); J, w, r=multipleCCA(Cxx, Cxy, Cyy, reg)

Copyright (c) MindAffect B.V. 2018

mindaffectBCI.decoder.multipleCCA.**robust_whitener**(*C: numpy.ndarray*, *reg: float = 0*, *rcond: float = 1e-06*, *symetric: bool = True*, *verb: int = 0*)

compute a robust whitener for the input covariance matrix C, s.t. isqrtC*C*isqrtC.T = I :param C: Sample covariance matrix of the data :type C: (d,d) np.ndarray :param reg: regularization strength when computing the whitener. Defaults to 0. :type reg: float, optional :param rcond: reverse-condition number for truncating degenerate eigen-values when computing the inverse. Defaults to 1e-6. :type rcond: float, optional :param symetric: flag to produce a symetric-whitener (which preserves location) or not. Defaults to True. :type symetric: bool, optional :param verb: verbosity level. Defaults to 0. :type verb: int, optional

Returns The whitening matrix iW (np.ndarray): The inverse whitening matrix

Return type W (np.ndarray)

mindaffectBCI.decoder.multipleCCA.**testcase**()

mindaffectBCI.decoder.multipleCCA.**testcase_matlab_summarystatistics**()

mindaffectBCI.decoder.multipleMCCA module

mindaffectBCI.decoder.multipleMCCA.**cvSupervised**(*Xe*, *Me*, *stimTimes*, *evtlabs=('re'*, *'fe')*, *n_splits=10*, *rank=1*)

do a cross-validated training of a multicca model using sklearn

mindaffectBCI.decoder.multipleMCCA.**multipleCCA**(*Cxx=None*, *Cxy=None*, *Cyy=None*, *reg=0*, *regy=0*, *rank=1*, *CCA=True*, *rcond=1e-06*, *symetric=False*)

Compute multiple multi-way CCA (MCCA) decompositions using the given summary statistics
[J,W,R]=multiCCA(Cxx,Cxy,Cyy,regx,regy,rank,CCA)

Inputs: Cxx = (d,d) current data cross-covariance (or any n-way symetric covariance matrix) Cxy = (nM,nE,tau,d) current per output ERPs Cyy = (nM,nE,tau,nE,tau) current response covariance for each output reg = (1,) :float or (2,):float linear weighting reg strength or separate values for Cxx and Cyy

OR (d,) regularisation ridge coefficients for Cxx (0)

rank= [1x1] number of top cca components to return (1) CCA = [bool] or (2,):bool flag if we normalize the X component, or the Y component. (true) rcond = [float] tolerance for singular eigenvalues. (1e-4)

or (2,) separate rcond for Cxx (rcond(1)) and Cyy (rcond(2)) or (2,1) (-1<-0) negative values = keep this fraction of eigen-values or (2,1) <-1 keep this many eigenvalues

symetric = [bool] us symetric whitener?

Outputs: J = [nY x 1] optimisation objective scores W = [d x rank x nY] spatial filters for each output R = [tau x nE x rank x nY] responses for each stimulus event for each output

Examples

Supervised CCA Y = (nEp/nSamp,nY,nE) indicator for each event-type and output of it's type in each epoch X = (nEp/nSamp,tau,d) sliced pre-processed raw data into per-stimulus event responses stimTimes = (nEp) # sample number for each epoch or None if sample rate Cxx,Cxy,Cyy=updateSummaryStatistics(X, Y, stimTimes) J,w, r=multiCCA(Cxx, Cxy, Cyy, reg) # w=(nM,d)[dx1],r=(nM,nE,tau)[tau x nE] are the found spatial,spectral filters # incremental zeroTrain Cxx=[];Cxy=[];Cyy=[]; prediction=[]; while isCalibration:

Slicing newX,newE,stimTimes= preprocessnSliceTrial(); #function which get 1 trials data+stim # Model Fitting Cxx, Cxy, Cyy= updateSummaryStatistics(newX, newE, stimTimes, Cxx, Cxy, Cyy); J, w, r = multipleCCA(Cxx, Cxy, Cyy, regx, regy) # w=[dx1],r=[tau x nE] are the found spatial,spectral filters

end # re-compute with current state, can try different regulisors, e.g. for adaptive bad-ch regx = updateRegFromBadCh(); J, w, r=multipleCCA(Cxx, Cxy, Cyy, reg)

Copyright (c) MindAffect B.V. 2018

mindaffectBCI.decoder.multipleMCCA.**plot_multicca_solution**(*w*, *r*)

mindaffectBCI.decoder.multipleMCCA.**robust_whitener**(*C: numpy.ndarray*, *reg: float = 0*, *rcond: float = 1e-06*, *symetric: bool = True*, *verb: int = 0*)

compute a robust whitener for the input covariance matrix C, s.t. isqrtC*C*isqrtC.T = I :param C: Sample covariance matrix of the data :type C: (d,d) np.ndarray :param reg: regularization strength when computing the whitener. Defaults to 0. :type reg: float, optional :param rcond: reverse-condition number for truncating degenerate eigen-values when computing the inverse. Defaults to 1e-6. :type rcond: float, optional :param symetric: flag to produce a symetric-whitener (which preserves location) or not. Defaults to True. :type symetric: bool, optional :param verb: verbosity level. Defaults to 0. :type verb: int, optional

Returns The whitening matrix iW (np.ndarray): The inverse whitening matrix

Return type W (np.ndarray)

mindaffectBCI.decoder.multipleMCCA.**testcase**()

mindaffectBCI.decoder.normalizeOutputScores module

mindaffectBCI.decoder.normalizeOutputScores.**c4**(*n*)

mindaffectBCI.decoder.normalizeOutputScores.**estimate_Fy_noise_variance**(*Fy*, *decisIdx=None*, *centFy=True*, *detrendFy=False*, *priorsigma=None*, *verb=0*)

Estimate the noise variance for Fy

Parameters

- **Fy** (*[np.ndarray]*) – (nTr,nEp,nY) the output scores
- **decisIdx** (*[type], optional*) – [description]. Defaults to None.
- **centFy** (*bool, optional*) – flag if we should center over outputs before computing variance. Defaults to True.
- **detrendFy** (*bool, optional*) – flag if we should detrend Fy before computing it's variance. Defaults to True.
- **priorsigma** (*[type], optional*) – Prior estimate for the variance. Defaults to None.

Returns (nTr, nDecis) estimated variance per sample at each decision point

Return type sigma2 ([np.ndarray])

mindaffectBCI.decoder.normalizeOutputScores.**estimate_Fy_noise_variance_2**(*Fy*, *decisIdx=None*, *centFy=True*, *detrendFy=False*, *priorsigma=None*, *verb=0*)

Estimate the noise variance for Fy

Parameters

- **Fy** (*[np.ndarray]*) – (nTr,nEp,nY) the output scores
- **decisIdx** (*[type], optional*) – [description]. Defaults to None.
- **centFy** (*bool, optional*) – flag if we should center over outputs before computing variance. Defaults to True.
- **detrendFy** (*bool, optional*) – flag if we should detrend Fy before computing it's variance. Defaults to True.
- **priorsigma** (*[type], optional*) – Prior estimate for the variance. Defaults to None.

Returns (nTr, nDecis) estimated variance per sample at each decision point

Return type sigma2 ([np.ndarray])

mindaffectBCI.decoder.normalizeOutputScores.**filter_Fy**(*Fy*, *filtLen=None*, *B=None*)

mindaffectBCI.decoder.normalizeOutputScores.**get_valid_epochs_outputs**(*Fy*, *validTgt=None*)

get the number valid epoch and outputs from a zero-padded Fy set

Parameters

- **Fy** (*[np.ndarray]*) – (nTrl,nEp,nY) the output scores
- **validTgt** (*[np.ndarray]*) – (nTrl,nY) flag if this output is used in this trial

Returns (nTrl,) number valid outputs per trial nEp ([np.ndarray]): (nTrl,) number valid epochs per trial validTgt ([np.ndarray]): (nTrl,nY) flag if this output is used in this trial

Return type nY ([np.ndarray])

mindaffectBCI.decoder.normalizeOutputScores.**mktestFy**(*nY=10*, *nM=1*, *nEp=360*, *nTrl=100*, *sigstr=0.5*, *startupNoisefrac=0.5*, *offsetstr=10*, *trlenfrac=0.25*)

mindaffectBCI.decoder.normalizeOutputScores.**normalizeOutputScores**(*Fy*, *validTgt=None*, *badFyThresh=4*, *centFy=True*, *detrendFy=False*, *nEpochCorrection=0*, *minDecisLen=0*, *maxDecisLen=0*, *bwdAccumulate=False*, *priorsigma=None*)

normalize the raw output scores to feed into the Perr computation

Parameters

- **Fy** (*nM,nTr,nEp,nY*) – float
- **validTgt** (*nM,nTr,nY*) – bool indication of which outputs are used in each trial
- **badFyThresh** – threshold for detction of bad Fy entry, in std-dev
- **centFy** (*bool*) – do we center Fy before computing *important* (true)
- **nEpochCorrection** (*int*) – number of epochs to use a base for correction of number epochs in the sum. basically, nEpoch < nEpochCorrection have highly increased Perr, so unlikely to be selected
- **minDecisLen** (*int*) –

  int, number of epochs to use as base for distribution of time-based decision points.
  i.e. decisions at, [1,2,4,...2^n]*exptDistDecis

  OR: minDecisLen<0 => decision point every abs(minDeicsLen) epochs
- **maxDecisLen** (*int*) – maximum number of epochs for a decision
- **bwdAccumulate** (*bool*) – accumulate data backwards from last epoch gathered
- **prior_sigma** (*float,float*) – prior estimate of the variance and it's equivalent samples weight (sigma,N)

Returns scaled summed scores sFy_scale (nM,nTr,nDecis,1): the normalization scaling factor Nep (nDecis,): number of epochs included in each score nEp,nY (nTrl): the detected number epoch/output for each trial

Return type ssFy (nM,nTr,nDecis,nY)

Copyright (c) MindAffect B.V. 2018

mindaffectBCI.decoder.normalizeOutputScores.**plot_normalizedScores**(*Fy*, *ssFy*, *scale_sFy*, *decisIdx*)

plot the normalized score and raw summed score for comparsion

mindaffectBCI.decoder.normalizeOutputScores.**testcase**()

mindaffectBCI.decoder.normalizeOutputScores_streamed module

mindaffectBCI.decoder.normalizeOutputScores_streamed.**compute_pval_curve**(*X*, *pval*)

mindaffectBCI.decoder.normalizeOutputScores_streamed.**compute_softmax_curve**(*X*, *pval*, *scale=3*)

mindaffectBCI.decoder.normalizeOutputScores_streamed.**incremental_estimate_noise_variance**(*Fy*, *ol*, *fil*

mindaffectBCI.decoder.normalizeOutputScores_streamed.**normalizeOutputScores_streamed**(*Fy*, *validTgt=*, *bad-FyThresh*, *centFy=T*, *block-Size=30*, *filtLen=5*

normalize the raw output scores to feed into the Perr computation Inputs:

Fy - (nM,nTr,nEp,nY):float validTgt - (nM,nTr,nY):bool indication of which outputs are used in each trial

badFyThresh - threshold for detction of bad Fy entry, in std-dev centFy - bool, do we center Fy before computing *important* (true) blockSize - maximum number of epochs single normalized sum

Outputs: ssFy - (nM,nTr,nDecis,nY):float scaled summed scores scale_sFy - (nM,nTr,nDecis):float scaling factor for each decision point N - (nDecis):int the number of elements summed in each block

Copyright (c) MindAffect B.V. 2018

mindaffectBCI.decoder.normalizeOutputScores_streamed.**softmax_nout_corr**(*n*)
approximate correction factor for probabilities out of soft-max to correct for number of outputs

mindaffectBCI.decoder.normalizeOutputScores_streamed.**softmax_vs_nout**(*nruns*, *nout*, *sscale*)

mindaffectBCI.decoder.normalizeOutputScores_streamed.**testcase**(*Fy=None*)

mindaffectBCI.decoder.offline_analysis module

mindaffectBCI.decoder.preprocess module

mindaffectBCI.decoder.preprocess.**butter_filterbank**(*X: numpy.ndarray*, *filterbank*, *fs: float*, *axis=-2*, *order=4*, *ftype='butter'*, *verb=1*)

mindaffectBCI.decoder.preprocess.**extract_envelope**(*X*, *fs*, *stopband=None*, *whiten=True*, *filterbank=None*, *log=True*, *env_stopband=(10, -1)*, *verb=False*, *plot=False*)

extract the envelope from the input data

Parameters

- **X** ([type]) – [description]
- **fs** ([type]) – [description]
- **stopband** ([type], optional) – pre-filter stop band. Defaults to None.
- **whiten** (bool, optional) – flag if we spatially whiten before envelope extraction. Defaults to True.
- **filterbank** ([type], optional) – set of filters to apply to extract the envelope for each filter output. Defaults to None.
- **log** (bool, optional) – flag if we return raw power or log-power. Defaults to True.
- **env_stopband** (tuple, optional) – post-filter on the extracted envelopes. Defaults to (10,-1).
- **verb** (bool, optional) – verbosity level. Defaults to False.
- **plot** (bool, optional) – flag if we plot the result of each preprocessing step. Defaults to False.

Returns the extracted envelopes

Return type X

mindaffectBCI.decoder.preprocess.**fft_filterbank**(*X: numpy.ndarray*, *filterbank*, *fs: float*, *axis=-2*, *verb=1*)

mindaffectBCI.decoder.preprocess.**fir**(*X: numpy.ndarray*, *ntap=3*, *dilation=1*)

mindaffectBCI.decoder.preprocess.**plot_grand_average_spectrum**(*X*, *fs: float*, *axis: int = -2*, *ch_names=None*, *log=False*)

mindaffectBCI.decoder.preprocess.**preprocess**(*X*, *Y*, *coords*, *fs=None*, *whiten=False*, *whiten_spectrum=False*, *decorrelate=False*, *badChannelThresh=None*, *badTrialThresh=None*, *center=False*, *car=False*, *standardize=False*, *stopband=None*, *filterbank=None*, *nY=None*, *fir=None*)

apply simple pre-processing to an input dataset

Parameters

- **X** ([type]) – the EEG data (tr,samp,d)
- **Y** ([type]) – the stimulus (tr,samp,e)
- **coords** ([type]) – [description]
- **whiten** (float, optional) – if >0 then strength of the spatially regularized whitener. Defaults to False.
- **whiten_spectrum** (float, optional) – if >0 then strength of the spectrally regularized whitener. Defaults to False.
- **badChannelThresh** ([type], optional) – threshold in standard deviations for detection and removal of bad channels. Defaults to None.
- **badTrialThresh** ([type], optional) – threshold in standard deviations for detection and removal of bad trials. Defaults to None.
- **center** (bool, optional) – flag if we should temporally center the data. Defaults to False.

- **car** (*bool, optional*) – flag if we should spatially common-average-reference the data. Defaults to False.

Returns the EEG data (tr,samp,d) Y ([type]): the stimulus (tr,samp,e) coords ([type]): meta-info for the data

Return type X ([type])

mindaffectBCI.decoder.preprocess.**rmBadChannels**(*X: numpy.ndarray*, *Y: numpy.ndarray*, *coords*, *thresh=3.5*)

remove bad channels from the input dataset

Parameters

- **X** (*[np.ndarray]*) – the eeg data as (trl,sample,channel)
- **Y** (*[np.ndarray]*) – the associated stimulus info as (trl,sample,stim)
- **coords** (*[type]*) – the meta-info about the channel
- **thresh** (*float, optional*) – threshold in standard-deviations for removal. Defaults to 3.5.

Returns X (np.ndarray) Y (np.ndarray) coords

mindaffectBCI.decoder.preprocess.**rmBadTrial**(*X*, *Y*, *coords*, *thresh=3.5*, *verb=1*)

[summary]

Parameters

- **X** (*[np.ndarray]*) – the eeg data as (trl,sample,channel)
- **Y** (*[np.ndarray]*) – the associated stimulus info as (trl,sample,stim)
- **coords** (*[type]*) – the meta-info about the channel
- **thresh** (*float, optional*) – threshold in standard-deviations for removal. Defaults to 3.5.

Returns X (np.ndarray) Y (np.ndarray) coords

mindaffectBCI.decoder.preprocess.**spatially_whiten**(*X: numpy.ndarray*, **args*, ***kwargs*)

spatially whiten the nd-array X

Parameters **X** (*np.ndarray*) – the data to be whitened, with channels/space in the *last* axis

Returns the whitened X W (np.ndarray): the whitening matrix used to whiten X

Return type X (np.ndarray)

mindaffectBCI.decoder.preprocess.**spectrally_whiten**(*X: numpy.ndarray*, *reg=0.01*, *axis=-2*)

spatially whiten the nd-array X

Parameters **X** (*np.ndarray*) – the data to be whitened, with channels/space in the *last* axis

Returns the whitened X W (np.ndarray): the whitening matrix used to whiten X

Return type X (np.ndarray)

mindaffectBCI.decoder.preprocess.**standardize_channel_power**(*X: numpy.ndarray*, *sigma2: numpy.ndarray = None*, *axis=-2*, *reg=0.1*, *alpha=0.001*)

Adaptively standardize the channel powers

Parameters

- **X** (*np.ndarray*) – The data to standardize
- **sigma2** (*np.ndarray, optional*) – previous channel powers estimates. Defaults to None.
- **axis** (*int, optional*) – dimension of X which is time. Defaults to -2.
- **reg** (*[type], optional*) – Regularisation strength for power estimation. Defaults to 1e-1.
- **alpha** (*[type], optional*) – learning rate for power estimation. Defaults to 1e-3.

Returns the standardized version of X sigma2 : the estimated channel power at the last sample of X

Return type sX

mindaffectBCI.decoder.preprocess.**temporally_decorrelate**(*X: numpy.ndarray, W: numpy.ndarray = 50, reg=0.5, eta=1e-07, axis=-2, verb=0*)

temporally decorrelate each channel of X by fitting and subtracting an AR model

Parameters

- **X** (*np.ndarray trl,samp,d*) – the data to be whitened, with channels/space in the *last* axis
- **W** (*tau,d*) – per channel AR coefficients
- **reg** (*float*) – regularization strength for fitting the AR model. Defaults to 1e-2
- **eta** (*float*) – learning rate for the SGD. Defaults to 1e-5

Returns the whitened X W (np.ndarray (tau,d)): the AR model used to sample ahead predict X

Return type X (np.ndarray)

mindaffectBCI.decoder.preprocess.**testCase_filterbank**()

mindaffectBCI.decoder.preprocess.**testCase_spectralwhiten**()

mindaffectBCI.decoder.preprocess.**testCase_temporallydecorrelate**(*X=None, fs=100*)

mindaffectBCI.decoder.preprocess.**test_fir**()

mindaffectBCI.decoder.readCapInf module

mindaffectBCI.decoder.readCapInf.**getPosInfo**(*chnames=None, capFile='1010', overridechnms=0, prefixMatch=False, verb=0, capDir=None*)

add electrode position info to a dimInfo structure

cnames,pos2d,pos3d,iseeg=getPosInfo(di,capFile,overridechnms,prefixMatch,verb,capDir)

Inputs: cnames – channel names to get pos-info for capFile – file name of a file which contains the pos-info for this cap overridechnms – flag that we should ignore the channel names in di prefixMatch – [bool] match channel names if only the start matches? verb – [int] verbosity level (0) capDir – 'str' directory to search for capFile

Outputs: cnames – [str] the names of the channels xy – (nCh,2):float the 2d position of the channels xyz – (nCh,2):float the 3d position of the channels iseeg - (nCh,):bool flag if this is an eeg channel, i.e. if it's name matched

mindaffectBCI.decoder.readCapInf.**latlong2xy**(*latlong*)
convert lat-long to 2-d unrolled x,y coordinates

mindaffectBCI.decoder.readCapInf.**latlong2xyz**(*latlong*)
convert lat-long into 3-d x,y,z coords on the sphere

mindaffectBCI.decoder.readCapInf.**readCapInf**(*cap='1010.txt'*, *capRoots=None*, *verb=0*)
read a cap file

Cname,latlong,xy,xyz,capfile=readCapInf(cap,capDir)

Inputs:

cap – file name of the cap-file capRoot – directory(s) to look for caps in (['.',mfiledir,mfiledir'/positions','../../resources/caps/'])

mindaffectBCI.decoder.readCapInf.**testCase**()

mindaffectBCI.decoder.readCapInf.**xy2latlong**(*xy*)
convert xy to lat-long, taking care to prevent division by 0

mindaffectBCI.decoder.readCapInf.**xyz2xy**(*xyz*)
convert 3d xyz coords to unrolled 2d xy positions

mindaffectBCI.decoder.scoreOutput module

mindaffectBCI.decoder.scoreOutput.**convWX**(*X*, *W*)
apply spatial filter W to X

mindaffectBCI.decoder.scoreOutput.**convXYR**(*X*, *Y*, *W*, *R*, *offset*)

mindaffectBCI.decoder.scoreOutput.**convYR**(*Y*, *R*, *offset=None*)
compute the convolution of Y with R

mindaffectBCI.decoder.scoreOutput.**datasettest**()

mindaffectBCI.decoder.scoreOutput.**dedupY0**(*Y*, *zerodup=True*, *yfeatdim=True*, *verb=0*)
remove outputs which are duplicates of the first (objID==0) output Inputs:

Y=(tr,ep,Y,e) zerodup : bool

if True, then if mi is the duplicate, then zero the duplicate, i.e. of Y[...,mi,:]=0 else, we zero out ojID==0, i.e. Y[...,0,:]=0

Outputs: Y=(tr,ep,Y,e) version of Y with duplicates of 1st row of Y set to 0

mindaffectBCI.decoder.scoreOutput.**plot_Fy**(*Fy*, *cumsum=True*, *label=None*, *legend=False*, *maxplots=25*)

mindaffectBCI.decoder.scoreOutput.**plot_Fycomparsion**(*Fy*, *Fys*, *ti=0*)

mindaffectBCI.decoder.scoreOutput.**plot_outputscore**(*X*, *Y*, *W=None*, *R=None*, *offset=0*)

mindaffectBCI.decoder.scoreOutput.**scoreOutput**(*Fe_mTSe*, *Y_TSye*, *dedup0=None*, *R=None*, *offset=None*, *outputscore='ip'*)
score each output given information on which stim-sequences corrospend to which inputs

Args

Fe_mTSe (nM,nTrl,nSamp,nE): similarity score for each event type for each stimulus Y_TSye (nTrl,nSamp,nY,nE): Indicator for which events occured for which outputs

nE=#event-types nY=#possible-outputs nEpoch=#stimulus events to process

R_mket (nM,nfilt,nE,tau): FWD-model (impulse response) for each of the event types, used to correct the scores for correlated responses.

offset (int): A (set of) offsets to try when decoding. Defaults to None. dedup0 (int): remove duplicate copies of output O, >0 remove the copy, <0 remove objID==0 (used when cross validating calibration data) outputscore (str): type of score to compute. one-of: 'ip', 'sse'. Defaults to 'ip'

Returns Fy_mTSy (nM,nTrl,nSamp,nY): similarity score for each input epoch for each output

Copyright (c) MindAffect B.V. 2018

mindaffectBCI.decoder.scoreOutput.**testcases**()

mindaffectBCI.decoder.scoreStimulus module

mindaffectBCI.decoder.scoreStimulus.**factored2full**(*W*, *R*)
convert a factored spatio-temporal model to a full model Inputs:

W (nM,rank,d) spatial filter set (BWD model) R (nM,rank,e,tau) temporal filter set (FWD model)

Output: W (nM,e,tau,d) spatio-temporal filter (BWD model)

mindaffectBCI.decoder.scoreStimulus.**plot_Fe**(*Fe*)

mindaffectBCI.decoder.scoreStimulus.**scoreStimulus**(*X*, *W*, *R=None*, *b=None*, *offset=0*, *f=None*, *isepoched=None*)
Apply spatio-temporal (possibly factored) model to data Inputs:

X = (nTrl x nSamp x d) raw response for the current stimulus event

d=#electrodes tau=#response-samples nEpoch=#stimulus events to process

OR (nTrl x nEpoch x tau x d) pre-sliced raw data

W = (nM x nfilt x d) spatial filters for each output R = (nM x nfilt x nE x tau) responses for each stimulus event for each output

OR W = (nM x nE x tau x d) spatio-temporal filter per event type and model R = None b = (nE,1) bias for each stimulus type offset = 0 (1,1) offset in X for applying W

Outputs: Fe= (nM x nTrl x nEpoch/nSamp x nE) similarity score for each input epoch for each output

Copyright (c) MindAffect B.V. 2018

mindaffectBCI.decoder.scoreStimulus.**scoreStimulusCont**(*X*, *W*, *R=None*, *b=None*, *offset=0*)
Apply spatio-tempoal (possibly factored) model to raw (non epoched) data

Parameters

- **X** (*np.ndarray (nTr,nSamp,d)*) – raw per-trial data
- **W** (*np.ndarray (nM,nfilt,d)*) – spatial filters for each factor

- **R** (*np.ndarray (nM,nfilt,nE,tau)*) – responses for each stimulus event for each output
- **b** (*np.ndarray (nE,1)*) – offset for each stimulus type

Returns similarity score for each input epoch for each output

Return type np.ndarray (nM,nTrl,nSamp,nE)

mindaffectBCI.decoder.scoreStimulus.**scoreStimulusEpoch**(*X*, *W*, *R=None*, *b=None*)

Apply spatio-temporal (possibly factored) model to epoched data

X = (nTrl x nEpoch x tau x d) pre-sliced raw data W = (nM x nfilt x d) spatial filters for each output R = (nM x nfilt x nE x tau) responses for each stimulus event for each output

OR W = (nM x nE x tau x d) spatio-temporal filter per event type and model R = None b = (nE,1) offset for each stimulus type

Outputs: Fe= (nM x nTrl x nEpoch/nSamp x nE) similarity score for each input epoch for each output

mindaffectBCI.decoder.scoreStimulus.**scoreStimulusEpoch_factored**(*X*, *W*, *R*, *b=None*)

Apply factored spatio-temporal model to epoched data X = (nTrl x nEpoch x tau x d) pre-sliced raw data W = (nM x nfilt x d) spatial filters for each output R = (nM x nfilt x nE x tau) responses for each stimulus event for each output b = (nE,1) offset for each stimulus type

Outputs: Fe= (nM x nTrl x nEpoch x nE) similarity score for each input epoch for each output

mindaffectBCI.decoder.scoreStimulus.**scoreStimulusEpoch_full**(*X*, *W*, *b=None*)

Apply full spatio-temporal model to epoched data X = (nTrl x nEpoch x tau x d) pre-sliced raw data W = (nM x nE x tau x d) spatio-temporal filter per event type and model b = (nE,1) offset for each stimulus type

Outputs: Fe= (nM x nTrl x nEpoch/nSamp x nE) similarity score for each input epoch for each output

mindaffectBCI.decoder.scoreStimulus.**testcase**()

mindaffectBCI.decoder.sigViewer module

mindaffectBCI.decoder.sigViewer.**parse_args**()

mindaffectBCI.decoder.sigViewer.**run**(*ui: mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface = None*, *host=None*, *timeout_ms: float = inf*, *center=True*, *timerange: int = 5*, *nstimulus_lines: int = 1*, *ndata_lines: int = -1*, *datastep: int = 20*, *stimstep: int = 1*, *stopband=((0, 3), (25, -1))*, *out_fs=60*, *ch_names=None*)

simple sig-viewer using the ring-buffer for testing

mindaffectBCI.decoder.sigViewer.**sigViewer**(**args*, ***kwargs*)

mindaffectBCI.decoder.startUtopiaHub module

mindaffectBCI.decoder.startUtopiaHub.**run**(*label=''*, *logdir=None*)

mindaffectBCI.decoder.stim2event module

mindaffectBCI.decoder.stim2event.**stim2event**(*M*, *evtypes=('re', 'fe')*, *axis=-1*, *oM=None*)

convert per-sample stimulus sequence into per-sample event sequence (e.g. rising/falling edge, or long/short flash)

Parameters

- **M** (*..samp) or (..,samp,nY*) – for and/non-target features
- **– [nE]** (*evnames*) – str list of strings: "0", "1", "00", "11", "01" (aka. 're'), "10" (aka, fe), "010" (aka. short), "0110" (aka long) "nt"+evtname : non-target event, i.e. evtname occured for any other target "any"+evtname: any event, i.e. evtname occured for *any* target "rest" - not any of the other event types, N.B. must be *last* in event list "raw" - unchanged input intensity coding "grad" - 1st temporal derivative of the raw intensity
- **–** (*oM*) –
- **–** –

Outputs: evt - (M.shape,nE) event sequence

Examples

#For a P300 bci, with target vs. non-target response use: M = np.array([[1,0,0,0,0,0,1,0],[0,0,1,0,0,0,0,0],[0,0,0,0,1,0,0,0]]).T E = stim2event(M,evtypes=['re','ntre'], axis=-2)

#Or modeling as two responses, target-stim-response and any-stim-response E = stim2event(M,evtypes=('re','anyre'), axis=-2)

mindaffectBCI.decoder.stim2event.**testcase**()

mindaffectBCI.decoder.timestamp_check module

mindaffectBCI.decoder.timestamp_check.**timestampPlot**(*filename=None*)

mindaffectBCI.decoder.trigger_check module

mindaffectBCI.decoder.trigger_check.**run**(*hostname='-'*, *stopband=(0.1*, *45*, *'bandpass')*, *fs_out=250*, ***kwargs*)

online continuously updating trigger check plot

mindaffectBCI.decoder.trigger_check.**triggerPlot**(*X*, *Y*, *fs*, *clsfr=None*, *fig=None*, *evtlabs=('re'*, *'fe')*, *tau_ms=125*, *offset_ms=-25*, *max_samp=10000*, *trntrl=None*, *plot_model=True*, *plot_trial=True*, *ax=None*, ***kwargs*)

mindaffectBCI.decoder.trigger_check.**trigger_check**(*filename=None*, *evtlabs=('re', 'fe')*, *tau_ms=125*, *offset_ms=-25*, *stopband=((0.1, 45, 'bandpass')*, *fs_out=250*, *trn-trl=slice(None, 10, None)*, *trn-samp=6000*, *max_samp=6000*, *plot_model=True*, *plot_trial=True*, *plot_epoch_lines=False*, ***kwargs*)

make a set of visualizations of the stimulus->measurement time-lock

Parameters

- **filename** (*[type], optional*) – The name of the file to load. Defaults to None.
- **evtlabs** (*tuple, optional*) – The event coding to use for how stimulus changes map into trigger inputs. Defaults to ('0','1').
- **tau_ms** (*int, optional*) – The duration of the stimulus response. Defaults to 400.
- **offset_ms** (*int, optional*) – Offset of the start of the stimulus response w.r.t. the trigger time. Defaults to -50.
- **max_samp** (*int, optional*) – Limit in the number of samples for each trial. Defaults to 6000.
- **stopband** (*tuple, optional*) – Temporal filter used to pre-process the EEG. Defaults to (.1,45,'bandpass').
- **fs_out** (*int, optional*) – Sample rate of the EEG used for analysis. Defaults to 250.

mindaffectBCI.decoder.updateSummaryStatistics module

mindaffectBCI.decoder.updateSummaryStatistics.**Cxx_diag2full**(*Cxx_tdd*)

mindaffectBCI.decoder.updateSummaryStatistics.**Cyx_diag2full**(*Cyx_tyed*, *tau*, *offset=None*)

mindaffectBCI.decoder.updateSummaryStatistics.**Cyy_diag2full**(*Cyy_tyee*)

convert diag compressed Cyy to full version

Parameters **Cyy** (*[type]*) – (tau,nY,nE,nY,nE) or (tau,nY,nE,nE)

Returns block up-scaled Cyy

Return type Cyy_yetyet (nY,nE,tau,nE,tau)

mindaffectBCI.decoder.updateSummaryStatistics.**Cyy_tyeye_diag2full**(*Cyy_tyeye*)

[summary]

Parameters **Cyy_tyeye** (*[type]*) – the input compressed version of Cyy

Returns the expanded version of Cyy

Return type Cyy_yetyet

mindaffectBCI.decoder.updateSummaryStatistics.**Cyy_yetet_diag2full**(*Cyy_yetet*)

[summary]

Parameters **Cyy_yetet** (*[type]*) – the input compressed version of Cyy

Returns the expanded version of Cyy

Return type Cyy_yetyet

mindaffectBCI.decoder.updateSummaryStatistics.**autocov**(*X*, *tau*)

Compute the cross-auto-correlation of X, i.e. spatial-temporal covariance Inputs:

X – (nTrl, nSamp, d) the data to have autocov computed nSamp = temporal dim, d = spatial dim

tau – number of samples in the autocov

Outputs: Ctdtd – (tau, d, tau, d)

mindaffectBCI.decoder.updateSummaryStatistics.**compCxx_diag**(*X_TSd*, *tau: float*, *offset: float = 0*, *unitnorm: bool = True*, *center: bool = False*)

Compute the main tau diagonal entries of a Cyy tensor :param X_TSd: the new stim info to be added

Indicator for which events occured for which outputs nE=#event-types nY=#possible-outputs nEpoch=#stimulus events to process

Parameters

- **tau** (*int*) – number of time-shifts to compute over
- **offset** (*int*) – offset in for tau=0 (Note: ignored here)
- **unitnorm** (*bool*) – flag if we normalize the Cyy with number epochs. Defaults to True.

Returns Cxx_tdd – (tau, d, d)

mindaffectBCI.decoder.updateSummaryStatistics.**compCxx_full**(*X_TSd*, *tau: float*, *offset=0*, *unitnorm: bool = False*, *center: bool = False*)

Compute the main tau diagonal entries of a Cyy tensor :param X_TSd: the input d dimensional data :type X_TSd: nTrl, nSamp, d :param tau: number of time-shifts to compute over :type tau: int :param offset: offset in for tau=0 (Note: ignored here) :type offset: int :param unitnorm: flag if we normalize the Cyy with number epochs. Defaults to True. :type unitnorm: bool

Returns Cxx_fdfd – (tau, d, tau d) the full cross auto-covariance

mindaffectBCI.decoder.updateSummaryStatistics.**compCyx_diag**(*X_TSd*, *Y_TSye*, *tau=None*, *offset=0*, *center=False*, *verb=0*, *unitnorm=True*)

Args: X_TSd (nTrl, nSamp, d): raw response for the current stimulus event

d=#electrodes

Y_TSye (nTrl, nSamp, nY, nE): Indicator for which events occured for which outputs nE=#event-types nY=#possible-outputs nEpoch=#stimulus events to process

Returns cross covariance at different offsets taus : relative offsets (t_y - t_x)

Return type Cyx_tyed (nY, nE, tau, d)

mindaffectBCI.decoder.updateSummaryStatistics.**compCyx_full**(*X_TSd*, *Y_TSye*, *tau=None*, *offset=0*, *center=False*, *verb=0*, *unitnorm=True*)

Args: X_TSd (nTrl, nSamp, d): raw response for the current stimulus event

d=#electrodes

Y_TSye (nTrl, nSamp, nY, nE): Indicator for which events occured for which outputs nE=#event-types nY=#possible-outputs nEpoch=#stimulus events to process

Returns cross covariance at different offsets

Return type Cyx_tyetd (nY, nE, tau, d)

mindaffectBCI.decoder.updateSummaryStatistics.**compCyy_diag**(*Y*, *tau: float*, *unitnorm: bool = True*)

Compute the main tau diagonal entries of a Cyy tensor :param Y: the new stim info to be added

Indicator for which events occured for which outputs nE=#event-types nY=#possible-outputs nEpoch=#stimulus events to process

Parameters

- **tau** (*int*) – number of samples in the stimulus response
- **unitnorm** (*bool*) – flag if we normalize the Cyy with number epochs. Defaults to True.

Returns

Return type Cyy_tyeye (tau,nY,nE,nY,nE)

mindaffectBCI.decoder.updateSummaryStatistics.**compCyy_diag_perY**(*Y*, *tau: float*, *unitnorm: bool = True*, *perY: bool = True*)

Compute the main tau diagonal entries of a Cyy tensor for each output independently :param Y_TSye: the new stim info to be added

Indicator for which events occured for which outputs nE=#event-types nY=#possible-outputs nEpoch=#stimulus events to process

Parameters

- **tau** (*int*) – number of samples in the stimulus response
- **unitnorm** (*bool*) – flag if we normalize the Cyy with number epochs. Defaults to True.

Returns

Return type Cyy_tyee (tau, nY, nE, nE)

mindaffectBCI.decoder.updateSummaryStatistics.**compCyy_full**(*Y_TSye*, *tau: int*, *offset: int = 0*, *unitnorm: bool = True*)

Compute the full YY cross auto-covariance

Parameters

- **Y_TSye** (*nTrl, nSamp, nY, nE*) – the new stim info to be added Indicator for which events occured for which outputs nE=#event-types nY=#possible-outputs nEpoch=#stimulus events to process
- **tau** (*int*) – number of samples in the stimulus response
- **offset** (*int*) – offset from time-zero for the cross computation (Note: ignored here)
- **unitnorm** (*bool*) – flag if we normalize the Cyy with number epochs. Defaults to True.

Returns Cyy_yetyet – (nY, nE, tau, nY, nE, tau)

mindaffectBCI.decoder.updateSummaryStatistics.**cov**(*X*)

Compute the spatial covariance Input:

X = (. . . ,d) data, with features in the last dimension

Output: Cxx = (d,d)

mindaffectBCI.decoder.updateSummaryStatistics.**crossautocov**(*X*, *Y*, *tau*, *offset=0*)

Compute the cross-auto-correlation between 2 datasets, X,Y, i.e. the double spatial-temporal covariance Inputs:

X – (nTrl, nSamp, dx) the data to have autocov computed nSamp = temporal dim, dx = spatial dim x

Y – (nTrl, nSamp, dy) the data to have autocov computed nSamp = temporal dim, dy = spatial dim y

tau – :int number of samples in the cov

OR [int,int] time lags for x and y

Outputs: Ctdtd – (taux, dx, tauy, dy)

mindaffectBCI.decoder.updateSummaryStatistics.**plotCxy**(*Cyx_yetd*, *evtlabs=None*, *fs=None*)

mindaffectBCI.decoder.updateSummaryStatistics.**plot_erp**(*erp*, *evtlabs=None*, *outputs=None*, *times=None*, *fs=None*, *ch_names=None*, *axis=-1*, *plottype='plot'*, *offset=0*, *ylim=None*, *suptitle: str = None*, *block: bool = False*)

Make a multi-plot of the event ERPs (as stored in erp) erp = (nY, nE, tau, d) current per output ERPs

mindaffectBCI.decoder.updateSummaryStatistics.**plot_factoredmodel**(*A*, *R*, *S=None*, *evtlabs=None*, *times=None*, *ch_names=None*, *ch_pos=None*, *fs=None*, *offset_ms=None*, *offset=None*, *spatial_filter_type='Filter'*, *label=None*, *ncol=2*)

Make a multi-plot of a factored model A_kd = k components and d sensors R_ket = k components, e events, tau samples response-duration

mindaffectBCI.decoder.updateSummaryStatistics.**plot_subspace**(*X_TSfd*, *Y_TSye*, *W_kd*, *R_ket*, *S_y*, *f_f*, *offset: int = 0*, *fs: float = 100*, *block: bool = False*, *label: str = None*)

plot the subspace source activity of a fwd-bwd model

Parameters

- **X_TSfd** (*[type]*) – [description]
- **Y_TSye** (*[type]*) – [description]
- **W_kd** (*[type]*) – [description]
- **R_ket** (*[type]*) – [description]
- **S_y** (*[type]*) – [description]
- **f_f** (*[type]*) – [description]
- **offset** (*int, optional*) – [description]. Defaults to 0.
- **fs** (*float, optional*) – [description]. Defaults to 100.
- **block** (*bool, optional*) – [description]. Defaults to False.
- **label** (*str, optional*) – [description]. Defaults to None.

mindaffectBCI.decoder.updateSummaryStatistics.**plot_summary_statistics**(*Cxx_dd*, *Cyx_yetd*, *Cyy_yetet*, *evtlabs=None*, *outputs=None*, *times=None*, *ch_names=None*, *fs=None*, *label: str = None*)

Visualize the summary statistics (Cxx_dd, Cyx_yetd, Cyy) of a dataset

It is assumed the data has 'd' channels, with 'nE' different types of trigger event, and a response length of 'tau' for each trigger.

Parameters

- **Cxx_dd** (*d, d*) – spatial covariance
- **Cxy_yetd** (*nY, nE, tau, d*) – per-output event related potentials (ERPs)
- **Cyy_yetet** (*nY, nE, tau, nE, tau*) – updated response covariance for each output
- **evtlabs** (*[type], optional*) – the labels for the event types. Defaults to None.
- **times** (*[type], optional*) – values for the time-points along tau. Defaults to None.
- **ch_names** (*[type], optional*) – textual names for the channels. Defaults to None.
- **fs** (*[type], optional*) – sampling rate for the data along tau (used to make times if not given). Defaults to None.

mindaffectBCI.decoder.updateSummaryStatistics.**plot_trial**(*X_TSd*, *Y_TSy*, *fs: float = None*, *ch_names=None*, *evtlabs=None*, *outputs=None*, *times=None*, *ylabel: str = 'ch + output'*, *suptitle: str = None*, *block: bool = False*)

visualize a single trial with data and stimulus sequences

Parameters

- **X_TSd** (*[type]*) – raw data sequence
- **Y_TSy** (*[type]*) – raw stimulus sequence
- **fs** (*float, optional*) – sample rate of data and stimulus. Defaults to None.
- **ch_names** (*list-of-str, optional*) – channel names for X. Defaults to None.

mindaffectBCI.decoder.updateSummaryStatistics.**testCases**()

mindaffectBCI.decoder.updateSummaryStatistics.**testComputationMethods**()

mindaffectBCI.decoder.updateSummaryStatistics.**testCyy2**()

mindaffectBCI.decoder.updateSummaryStatistics.**testSlicedvsContinuous**()

mindaffectBCI.decoder.updateSummaryStatistics.**test_compCxx_diag**()

mindaffectBCI.decoder.updateSummaryStatistics.**test_compCyx_diag**()

mindaffectBCI.decoder.updateSummaryStatistics.**updateCxx**(*Cxx*, *X*, *stimTimes=None*, *tau: int = None*, *wght: float = 1*, *offset: int = 0*, *center: bool = False*, *unitnorm: bool = True*)

Parameters

- **Cxx_dd** (*ndarray (d,d)*) – current data covariance
- **X_TSd** (*nTrl, nSamp, d*) – raw response at sample rate
- **stimTimes_samp** (*ndarray (nTrl, nEp)*) – sample times for start each epoch. Used to detect
- **wght** (*float*) – weight to accumulate this Cxx with the previous data, s.t. Cxx = Cxx_old*wght + Cxx_new. Defaults to 1.
- **center** (*bool*) – flag if center the data before computing covariance? Defaults to True.

Returns current data covariance

Return type Cxx_dd (ndarray (d,d))

mindaffectBCI.decoder.updateSummaryStatistics.**updateCxy**(*Cxy*, *X*, *Y*, *stimTimes=None*, *tau=None*, *wght=1*, *offset=0*, *center=False*, *verb=0*, *unitnorm=True*)

Parameters

- **X_TSd** (*nTrl, nSamp, d*) – raw response at sample rate
- **Y_TSye** (*nTrl, nSamp, nY, nE*) – event indicator at sample rate
- **=** (*stimTimes_samp*) – overlapping responses
- **Cxy_yetd** (*nY, nE, tau, d*) – current per output ERPs

Returns current per output ERPs

Return type Cxy_yetd (nY, nE, tau, d)

mindaffectBCI.decoder.updateSummaryStatistics.**updateCyy**(*Cyy*, *Y*, *stimTime=None*, *tau=None*, *wght=1*, *offset: float = 0*, *zeropadded=True*, *unitnorm=True*, *perY=True*)

Compute the Cyy tensors given new data :param Cyy_yetet: old Cyy info :param Y_TSye: the new stim info to be added

nE=#event-types nY=#possible-outputs nEpoch=#stimulus events to process

Parameters

- **tau** (*int*) – number of samples in the stimulus response
- **zeropadded** (*bool*) – flag variable length Y are padded with 0s
- **wght** (*float*) – weighting for the new vs. old data
- **unitnorm** (*bool*) – flag if we normalize the Cyy with number epochs

Returns

Return type Cyy_yetet (nY, tau, nE, nE)

mindaffectBCI.decoder.updateSummaryStatistics.**updateSummaryStatistics**(*X*, *Y*, *stimTimes=None*, *Cxx=None*, *Cxy=None*, *Cyy=None*, *badEpThresh=4*, *halflife_samp=1*, *cxxp=True*, *cyyp=True*, *tau=None*, *offset=0*, *center=True*, *unitnorm=True*, *zeropadded: bool = True*, *perY=True*)

Compute updated summary statistics (Cxx_dd, Cxy_yetd, Cyy_yetet) for new data in X with event-info Y

Parameters

- **X_TSd** (*nTrl, nEp, tau, d*) –

  raw response for the current stimulus event d=#electrodes tau=#response-samples nEpoch=#stimulus events to process

  OR (nTrl, nSamp, d)

- **=** (*stimTimes_samp*) – Indicator for which events occured for which outputs OR

(nTrl, nSamp, nY, nE) nE=#event-types nY=#possible-outputs nEpoch=#stimulus events to process

- **=** – overlapping responses
- **= int - the length of the impulse response in samples** (*tau*) –
- **offset** – int = relative shift of Y w.r.t. X
- **Cxx_dd** (*d, d*) – current data covariance
- **Cxy_yetd** (*nY, nE, tau, d*) – current per output ERPs
- **Cyy_yetet** (*nY, nE, tau, nE, tau*) – current response covariance for each output
- **badEpThresh** (*float*) – threshold for removing bad-data before fitting the summary statistics
- **center** (*bool*) – do we center the X data before computing the summary statistics? (True)
- **halflife_samp** (*float*) – forgetting factor for the updates Note: alpha = exp(log(.5)./(half-life)), half-life = log(.5)/log(alpha)

Returns updated data covariance Cxy_yetd (nY, nE, tau, d) : updated per output ERPs Cyy_yetet (nY, nE, tau, nE, tau): updated response covariance for each output

Return type Cxx_dd (d,d)

Examples

Supervised CCA Y = (nEpoch/nSamp, nY, nE) [nE x nY x nEpoch] indicator for each event-type and output of it's type in each epoch X = (nEpoch/nSamp, tau, d) [d x tau x nEpoch] sliced pre-processed raw data into per-stimulus event responses stimTimes = [nEpoch] sample numbers of the stimulus events

OR None if X, Y are samples

[Cxx, Cxy, Cyy] = updateSummaryStatistics(X, Y, 3*np.arange(X.shape[0])); [J, w, r]=multipleCCA(Cxx, Cxy, Cyy)

mindaffectBCI.decoder.utils module

class mindaffectBCI.decoder.utils.**RingBuffer**(*maxsize*, *shape*, *dtype=<class 'numpy.float32'>*)

Bases: object

time efficient linear ring-buffer for storing packed data, e.g. continguous np-arrays

append(*x*)
add single element to the ring buffer

clear()
empty the ring-buffer and reset to empty

extend(*x*)
add a group of elements to the ring buffer

shape

unwrap()
get a view on the valid portion of the ring buffer

mindaffectBCI.decoder.utils.**block_randomize**(*true_target*, *npermute*, *axis=-3*, *block_size=None*)
make a block random permutaton of the input array Inputs:

npermute: int - number permutations to make true_target: (..., nEp, nY, e): true target value for nTrl trials of length nEp flashes axis : int the axis along which to permute true_target

mindaffectBCI.decoder.utils.**butter_sosfilt**(*X*, *stopband*, *fs: float*, *order: int = 6*, *axis: int = -2*, *zi=None*, *verb=True*, *ftype='butter'*)
use a (cascade of) butterworth SOS filter(s) filter X along axis

Parameters

- **X** (*np.ndarray*) – the data to be filtered
- **stopband** (*[type]*) – the filter band specifications in Hz, as a list of lists of stopbands (given as (low-pass,high-pass)) or pass bands (given as (low-cut,high-cut,'bandpass'))
- **fs** (*float*) – the sampling rate of X
- **order** (*int, optional*) – the desired filter order. Defaults to 6.
- **axis** (*int, optional*) – the axis of X to filter along. Defaults to -2.
- **zi** (*[type], optional*) – the internal filter state – propogate between calls for incremental filtering. Defaults to None.
- **verb** (*bool, optional*) – Verbosity level for logging. Defaults to True.
- **ftype** (*str, optional*) – The type of filter to make, one-of: 'butter', 'bessel'. Defaults to 'butter'.

Returns the filtered version of X sos (np.ndarray): the designed filter coefficients zi (np.ndarray): the filter state for propogation between calls

Return type X [np.ndarray]

mindaffectBCI.decoder.utils.**equals_subarray**(*a*, *pat*, *axis=-1*, *match=-1*)
efficiently find matches of a 1-d sub-array along axis within an nd-array

mindaffectBCI.decoder.utils.**extract_ringbuffer_segment**(*rb*, *bgn_ts*, *end_ts=None*)
extract the data between start/end time stamps, from time-stamps contained in the last channel of a nd matrix

mindaffectBCI.decoder.utils.**idOutliers**(*X*, *thresh=4*, *axis=-2*, *verbosity=0*)
identify outliers with excessively high power in the input data Inputs:

X:float the data to identify outliers in axis:int (-2) axis of X to sum to get power thresh(float): threshold standard deviation for outlier detection verbosity(int): verbosity level

Returns bool (X.shape axis==1) indicator for outlying elements epPower:float (X.shape axis==1) power used to identify bad

Return type badEp

mindaffectBCI.decoder.utils.**iir_sosfilt_sos**(*stopband*, *fs*, *order=4*, *ftype='butter'*, *passband=None*, *verb=0*)
given a set of filter cutoffs return butterworth or bessel sos coefficients

mindaffectBCI.decoder.utils.**lab2ind**(*lab*, *lab2class=None*)
convert a list of labels (as integers) to a class indicator matrix

mindaffectBCI.decoder.utils.**randomSummaryStats**(*d=10*, *nE=2*, *tau=10*, *nY=1*)

mindaffectBCI.decoder.utils.**robust_mean**(*X*, *thresh=(3, 3)*)
Compute robust mean of values in X, using gaussian outlier criteria

Parameters

- **X** (*the data*) – the data
- **thresh** (*2,*) – lower and upper threshold in standard deviations

Returns the robust mean good (): the indices of the 'good' data in X

Return type mu ()

mindaffectBCI.decoder.utils.**save_butter_sosfilt_coeff**(*filename=None*, *stopband=((45, 65), (5.5, 25, 'bandpass'))*, *fs=200*, *order=6*, *ftype='butter'*)

design a butterworth sos filter cascade and save the coefficients

mindaffectBCI.decoder.utils.**search_directories_for_file**(*f*, **args*)

search a given set of directories for given filename, return 1st match

Parameters

- **f** (*str*) – filename to search for (or a pattern)
- **()** (**args*) – set for directory names to look in

Returns the *first* full path to where f is found, or f if not found.

Return type f (str)

mindaffectBCI.decoder.utils.**sliceData**(*X*, *stimTimes_samp*, *tau=10*)

mindaffectBCI.decoder.utils.**sliceY**(*Y*, *stimTimes_samp*, *featdim=True*)

Y = (nTrl, nSamp, nY, nE) if featdim=True OR Y=(nTrl, nSamp, nY) if featdim=False #(nE x nY x nSamp x nTrl)

mindaffectBCI.decoder.utils.**sosfilt_2d_py**(*sos*, *X*, *axis=-2*, *zi=None*)

pure python fallback for second-order-sections filter in case scipy isn't available

mindaffectBCI.decoder.utils.**sosfilt_zi_py**(*sos*)

compute an initial state for a second-order section filter

mindaffectBCI.decoder.utils.**sosfilt_zi_warmup**(*zi*, *X*, *axis=-1*, *sos=None*)

Use some initial data to "warmup" a second-order-sections filter to reduce startup artifacts.

Parameters

- **zi** (*np.ndarray*) – the sos filter, state
- **X** (*[type]*) – the warmup data
- **axis** (*int, optional*) – The filter axis in X. Defaults to -1.
- **sos** (*[type], optional*) – the sos filter coefficients. Defaults to None.

Returns the warmed up filter coefficients

Return type [np.ndarray]

mindaffectBCI.decoder.utils.**testNoSignal**(*d=10*, *nE=2*, *nY=1*, *isi=5*, *tau=None*, *nSamp=10000*, *nTrl=1*)

mindaffectBCI.decoder.utils.**testSignal**(*nTrl=1*, *d=5*, *nE=2*, *nY=30*, *isi=5*, *tau=None*, *offset=0*, *nSamp=10000*, *stimthresh=0.6*, *noise2signal=1*, *irf=None*)

mindaffectBCI.decoder.utils.**test_butter_sosfilt**()

mindaffectBCI.decoder.utils.**test_sosfilt_py**()

mindaffectBCI.decoder.utils.**testtestSignal**()

mindaffectBCI.decoder.utils.**unwrap**(*x*, *range=None*)

unwrap a list of numbers to correct for truncation due to limited bit-resolution, e.g. time-stamps stored in 24bit integers

mindaffectBCI.decoder.utils.**unwrap_test**()

mindaffectBCI.decoder.utils.**upsample_codebook**(*trlen*, *cb*, *ep_idx*, *stim_dur_samp*, *offset_samp=(0, 0)*)

upsample a codebook definition to sample rate Inputs:

trlen : (int) length after up-sampling cb : (nTr, nEp, ...) the codebook ep_idx : (nTr, nEp) the indices of the codebook entries stim_dur_samp: (int) the amount of time the cb entry is held for offset_samp : (2,):int the offset for the stimulus in the upsampled trlen data

Outputs: Y : (nTrl, trlen, ...) the up-sampled codebook

mindaffectBCI.decoder.utils.**window_axis**(*a*, *winsz*, *axis=0*, *step=1*, *prependwindowdim=False*)

efficient view-based slicing of equal-sized equally-spaced windows along a selected axis of a numpy nd-array

mindaffectBCI.decoder.utils.**zero_outliers**(*X*, *Y*, *badEpThresh=4*, *badEpChThresh=None*, *verbosity=0*)

identify and zero-out bad/outlying data

Inputs: X = (nTrl, nSamp, d) Y = (nTrl, nSamp, nY, nE) OR (nTrl, nSamp, nE)

nE=#event-types nY=#possible-outputs nEpoch=#stimulus events to process

mindaffectBCI.decoder.zscore2Ptgt_softmax module

mindaffectBCI.decoder.zscore2Ptgt_softmax.**calibrate_softmaxscale**(*f*, *validTgt=None*, *scales=(0.01*, *0.02*, *0.05*, *0.1*, *0.2*, *0.3*, *0.4*, *0.5*, *1*, *1.5*, *2*, *2.5*, *3*, *3.5*, *4*, *5*, *7*, *10*, *15*, *20*, *30)*, *MINP=0.01*, *marginalizemodels=True*, *marginalizedecis=False*, *eta=0.05*, *error_weight=2*)

attempt to calibrate the scale for a softmax decoder to return calibrated probabilities

Parameters

- **f** (*(nM,)nTrl,nDecis,nY*) – normalized accumulated scores]
- **validTgt** (*bool (nM,)nTrl,nY*) – which targets are valid in which trials

- **(list** (*scales*) – int): set of possible soft-max scales to try
- **MINP** (*float*) – minimium P-value. We clip the true-target p-val to this level as a way of forcing the fit to concentrate on getting the p-val right when high, rather than over penalizing when it's wrong

Returns slope for softmax to return calibrated probabilities

Return type softmaxscale (float)

mindaffectBCI.decoder.zscore2Ptgt_softmax.**entropy**(*p*, *axis=-1*)

mindaffectBCI.decoder.zscore2Ptgt_softmax.**marginalize_scores**(*f*, *axis*, *prior=None*, *keepdims=False*)

marginalize the output scores to remove nuisance parameters, e.g. decis-pts, models.

Parameters

- **f** (*np.ndarray (nModel,nTrial,nDecisPts,nOutput)*) – the scores
- **axis** (*[listInt]*) – the axis of f to marginalize over
- **prior** (*[type], optional*) – prior over the dimesions of f. Defaults to None.
- **keepdims** (*bool, optional*) – flag if we keep or compress the dims of f. Defaults to False.

Returns (np.ndarray): the marginalized f scores

Return type f

mindaffectBCI.decoder.zscore2Ptgt_softmax.**mkTestFy**(*nY*, *nM*, *nEp*, *nTrl*, *sigstr*, *startup_lag*)

mindaffectBCI.decoder.zscore2Ptgt_softmax.**softmax**(*f*, *axis=-1*, *validTgt=None*)

simple softmax over final dim of input array, with compensation for missing inputs with validTgt mask.

mindaffectBCI.decoder.zscore2Ptgt_softmax.**softmax_nout_corr**(*n*)

approximate correction factor for probabilities out of soft-max to correct for number of outputs

mindaffectBCI.decoder.zscore2Ptgt_softmax.**testcase**(*nY=10*, *nM=4*, *nEp=340*, *nTrl=500*, *sigstr=0.4*, *normSum=True*, *centFy=True*, *detrendFy=True*, *marginalizemodels=True*, *marginalizedecis=False*, *bwdAccumulate=False*, *nEpochCorrection=20*, *priorweight=100.0*, *startup_lag=0.1*)

mindaffectBCI.decoder.zscore2Ptgt_softmax.**visPtgt**(*Fy*, *normSum*, *centFy*, *detrendFy*, *bwdAccumulate*, *marginalizemodels*, *marginalizedecis*, *nEpochCorrection*, *priorweight*, *minDecisLen=-1*)

mindaffectBCI.decoder.zscore2Ptgt_softmax.**zscore2Ptgt_softmax**(*f*, *softmaxscale: float = 2*, *prior: numpy.ndarray = None*, *validTgt=None*, *marginalizemodels: bool = True*, *marginalizedecis: bool = False*, *peroutputmodel: bool = True*)

convert normalized output scores into target probabilities

Parameters

- **f** (*nM,nTrl,nDecis,nY*) – normalized accumulated scores
- **softmaxscale** (*float, optional*) – slope to scale from scores to probabilities. Defaults to 2.
- **validtgtTrl** (*bool nM,nTrl,nY*) – which targets are valid in which trials
- **peroutputmode** (*bool, optional*) – assume if nM==nY that we have 1 model per output. Defaults to True.
- **prior** (*[nM,nTrl,nDecis,nY]*) – prior probabilities over the different dimesions of f. e.g. if want a prior over Models should be shape (nM,1,1,1), for a prior over outputs (nY,). Defaults to None.

Returns target probability for each trial (if marginalizemodels and marginalizedecis is True)

Return type Ptgt (nTrl,nY)

Module contents

Submodules

mindaffectBCI.noisetag module

class mindaffectBCI.noisetag.**CalibrationPhase**(*objIDs: int = 8*, *stimSeq=None*, *nTrials: int = 10*, *utopiaController=None*, *stimulusStateStack: mindaffectBCI.noisetag.GSM = None*, **args*, ***kwargs*)

Bases: *mindaffectBCI.noisetag.FSM*

do a complete calibration phase with nTrials x CalibrationTrial

Raises

- ValueError – [description]
- ValueError – [description]
- StopIteration – [description]

next(*t*)

get the next state in the sequence, moving through nTrials calibration trials where each trial has single trail stages of cue->wait->flicker

Parameters **t** (*int*) – current time

Raises StopIteration – when the whole sequence is complete

class mindaffectBCI.noisetag.**Experiment**(*objIDs*, *stimSeq=None*, *nCal=10*, *nPred=20*, *selectionThreshold=0.1*, *cuedprediction=False*, *utopiaController=None*, *stimulusStateStack=None*, *numframes=240.0*, *calframes=None*, *predframes=None*, *interphaseframes=900.0*, **args*, ***kwargs*)

Bases: *mindaffectBCI.noisetag.FSM*

do a complete experiment, with calibration -> prediction

next(*t*)

get the next state in the sequence, moving through calibration->prediction phases where each trial has single trail stages of cue->wait->flicker->feedback

Parameters **t** (*int*) – current time

Raises StopIteration – when the whole sequence is complete

class mindaffectBCI.noisetag.**FSM**

Bases: object

simple finite state machine, using a generator-like pattern

get()

get the current state, for this set of active objects

Returns

The current display state information as a 4-tuple with structure: (stimState, target_idx, objIDs, sendEvent) stimState (list-int): the stimulus state for each object as an integer target_idx (int): the index into stimState of the cued target, or -1 if no target set objIDs (list-int): the objectIDs for each of the outputs sendEvent (bool): flag if we should send stimulus events in this state

Return type tuple

next(*t*)

update the current state, return the new state, raise StopIteration exception when done

class mindaffectBCI.noisetag.**Flicker**(*stimSeq=None*, *numframes: int = 0.06666666666666667*, *tgtidx: int = -1*, *sendEvents: bool = True*, *framesperbit: int = 1*, *permute: bool = False*)

Bases: *mindaffectBCI.noisetag.FSM*

do a normal flicker sequence

get()

return the curent stimulus state info

Returns the current stimlus state tuple (stimState, target_idx, objIDs, sendEvent)

Return type StimulusState

next(*t*)

move to the next state

update_codebook_permutation()

update the permutation randomly mapping between codebook rows and outputs

update_ss()

update the information about the current stimulus state

class mindaffectBCI.noisetag.**FlickerWithSelection**(*stimSeq=None*, *numframes=0.06666666666666667*, *tgtidx=-1*, *utopiaController=None*, *framesperbit=1*, *sendEvents=True*, *permute=False*)

Bases: *mindaffectBCI.noisetag.Flicker*

do a normal flicker sequence, with early stopping selection

next(*t*)

get the next stimulus state in the sequence – terminate if out of time, or BCI made a selection

Parameters **t** (*int*) – current time

Raises StopIteration – stop if numframes exceeded or BCI made a selection

class mindaffectBCI.noisetag.**GSM**

Bases: *mindaffectBCI.noisetag.FSM*

Generalized state machine with stack of states

clear()

clear the state machine stack

get()

return the current stimulus state

Returns the current stimulus state tuple (stimState,target_idx,objIDs,sendEvent)

Return type StimulusState

next(*t*)

get the next stimulus state to shown

Parameters **t** (*int*) – the current time

Raises StopIteration – when this state machine has run out of states

pop()

remove and return the currently active state machine

Returns the current state machine at the stop of the stack

Return type *GSM*

push(*s*)

add a new state machine to the stack

Parameters **s** (GSM) – finite-state-machine to add

Returns current state machine stack

Return type list

class mindaffectBCI.noisetag.**HighlightObject**(*numframes=0.03333333333333333*, *tgtidx=-1*, *tgtState=2*, *sendEvents=False*, *numblinkframes=30*)

Bases: *mindaffectBCI.noisetag.Flicker*

'Highlight a single object for a number of frames

class mindaffectBCI.noisetag.**Noisetag**(*stimFile=None*, *utopiaController=None*, *stimulusStateMachineStack: mindaffectBCI.noisetag.GSM = None*, *clientid: str = None*)

Bases: object

noisetag abstraction layer to handle *both* the sequencing of the stimulus flicker, *and* the communications with the Mindaffect decoder. Clients can use this class to implement BCI control by:

0) setting the flicker sequence to use (method: startFlicker, startFlickerWithSelection, startCalibration, startPrediction, startExpt)

1) getting the current stimulus state (method: getStimulusState), and using that to draw the display

2) telling Noisetag when *exactly* the stimulus update took place (method: sendStimulusState)

3) getting the predictions/selections from noisetag and acting on them. (method: getLastPrediction() or getLastSelection())

addMessageHandler(*cb*)
add a handler which is called back when a new message is recieved

Parameters **cb** (*function*) – the function to be called for each newly recieved message

addPredictionHandler(*cb*)
add a handler which is called back when a Prediction is recieved from the decoder/hub

Parameters **cb** (*function*) – the function to be called for each newly recieved Prediction

addSelectionHandler(*cb*)
add a handler which is called back when a Selection is recieved from the decoder/hub

Parameters **cb** (*function*) – the function to be called for every newly recieved Selection

addSubscription(*msgs*)
add a set of messageIDs to our current set of subscribed message types.

Parameters **msgs** (*str*) – a list of messageIDs to subscribe to. See mindaffectBCI.utopiaclient for the list of message types and IDs

clearLastPrediction()
clear the information about the last recieved target prediction

clearLastSelection()
clear the last selelction message from the hub/decoder

clearLastSignalQuality()
clear the last signal quality message

connect(*host: str = None*, *port: int = -1*, *queryifhostnotfound: bool = True*, *timeout_ms: int = 5000*)
connect to the utopia hub

Parameters

- **host** (*str, optional*) – the hub hostname or ip address. Defaults to None.
- **port** (*int, optional*) – the hub port. Defaults to -1.
- **queryifhostnotfound** (*bool, optional*) – if auto-discovery fails do we query the user for the host IP. Defaults to True.
- **timeout_ms** (*int, optional*) – timeout in milliseconds for host autodiscovery or connection. Defaults to 5000.

Returns are we currently connected to the hub

Return type bool

getLastPrediction()
get the last prediction recieved from the hub/decoder

Returns the last recieved PredictedTargetProb message

Return type *PredictedTargetProb*

getLastSelection()
return the last selection message recieved from the hub/decoder

Returns the last Selection message recieved from the hub/decoder

Return type *Selection*

getLastSignalQuality()
return the last signal quality message recieved from the hub/decoder

Returns the last ElectrodeQuality message from the hub/decoder

Return type ElectrodeQuality

getNewMessages()
get all new messages from the decoder

Returns a list of all new UtopiaMessages recieved from the hub/decoder

Return type list-of-UtopiaMessage

getStimulusState(*objIDs=None*)
return the current stimlus state

Parameters **objIDs** (*list-of-int, optional*) – the set of objectIDs to get the state information for. Defaults to None.

Returns the current stimulus state tuple (stimState,target_idx,objIDs,sendEvent)

Return type StimulusState

getTimeStamp()
get the current time stamp

Returns the timestamp for the curren time in milliseconds

Return type int

gethostport()
return the hostname:port we are currently connected to

Returns the hub host:port we are currently connected to

Return type str

isConnected()
query the hub connection status

Returns are we connected to the hub

Return type bool

log(*msg*)
send a Log message to the decoder/hub

Parameters **msg** (*str*) – the Log message to send

modeChange(*newmode*)
manually change the decoder mode

Parameters **newmode** (*str*) – the new mode string to send to the hub/decoder

removeSubscription(*msgs*)
remove a set of messageIDs to our current set of subscribed message types.

Parameters **msgs** (*str*) – a list of messageIDs to unsubscribe from. See mindaffectBCI.utopiaclient for the list of message types and IDs

sendStimulusState(*timestamp=None*)
send the current stimulus state information to the decoder

Parameters **timestamp** (*int, optional*) – timestamp to use when sending the stimulus state information. Defaults to None.

setActiveObjIDs(*objIDs*)
update the set of active objects we send info to decoder about

Parameters **objIDs** (*list-of-int*) – the set of objectIDs to register

Returns the set of active object IDs

Return type list-of-int

setTimeStampClock(*tsclock*)
set the clock used by default to timestamp messages sent to the hub/decoder

Parameters **tsclock** (TimeStampClock) – the time-stamp clock object to use

set_isi(*new_isi: float*)

setnumActiveObjIDs(*nobj: int*)
update to say number active objects

Parameters **nobj** (*int*) – the number of active objects to set

Returns the current set of active object IDs

Return type list-of-int

startCalibration(*nTrials=10*, *stimSeq=None*, **args*, ***kwargs*)
setup and run a complete calibration phase

Parameters

- **nTrials** (*int, optional*) – number of calibration trials to run. Defaults to 10.
- **stimSeq** (*list-of-lists-of-int, optional*) – the stimulus sequence to play for the flicker in the calibration phase. Defaults to None.

startExpt(*nCal=1*, *nPred=20*, *selectionThreshold=0.1*, *cuedprediction=False*, **args*, ***kwargs*)
Start the sequence for a full Calibration->Prediction experiment.

Parameters

- **nCal** (*int, optional*) – the number of calibration trials. Defaults to 10.
- **nPred** (*int, optional*) – the number of prediction trials. Defaults to 20.
- **selectionThreshold** (*float, optional*) – the Perr threshold for selection. Defaults to .1
- **cuedprediction** (*bool, optional*) – flag if we do cueing before trial starts. Default to False.

startFlicker(*numframes=100*, *tgtidx=-1*, **args*, ***kwargs*)
setup and run the just the flicker

Parameters

- **numframes** (*int, optional*) – the number of frames for the flicker. Defaults to 100.

- **tgtidx** (*int, optional*) – the index in the stimSequence of the target, -1 if no cued target. Default to -1

startFlickerWithSelection (*numframes: int = 100*, *tgtidx: int = -1*, **args*, ***kwargs*)
setup and run the just the flicker, with early stopping if the BCI selects an output

Parameters

- **numframes** (*int, optional*) – the number of frames for the flicker. Defaults to 100.
- **tgtidx** (*int, optional*) – the index in the stimSequence of the target, -1 if no cued target. Default to -1

startPrediction (*nTrials=10*, *stimSeq=None*, **args*, ***kwargs*)
setup and run a complete prediction phase

Parameters

- **nTrials** (*int, optional*) – number of prediction trials to run. Defaults to 10.
- **stimSeq** (*list-of-lists-of-int, optional*) – the stimulus sequence to play for the flicker in the calibration phase. Defaults to None.

startSingleTrial (*numframes: int = 100*, *tgtidx: int = -1*, **args*, ***kwargs*)
setup and run a single flicker trial, with (optional)cue->wait->flicker->feedback

Parameters

- **numframes** (*int, optional*) – the number of frames for the flicker. Defaults to 100.
- **tgtidx** (*int, optional*) – the index in the stimSequence of the target, -1 if no cued target. Default to -1

subscribe (*msgs*)
tell the hub we will subscribe to this set of message IDs

Parameters **msgs** (*str*) – a list of messageIDs to subscribe to. See mindaffectBCI.utopiaclient for the list of message types and IDs

updateStimulusState (*t=None*)
update to the next stimulus state from the current sequence

Parameters **t** (*int, optional*) – the current time. Defaults to None.

class mindaffectBCI.noisetag.**PredictionPhase** (*objIDs*, *stimSeq=None*, *nTrials=10*, *utopiaController=None*, *stimulusStateStack=None*, *selectionThreshold=0.1*, *cuedprediction=False*, **args*, ***kwargs*)
Bases: *mindaffectBCI.noisetag.FSM*

do complete prediction phase with nTrials x trials with early-stopping feedback

Parameters **FSM** (*[type]*) – [description]

next (*t*)
get the next state in the sequence, moving through nTrials calibration trials where each trial has single trail stages of cue->wait->flicker->feedback

Parameters **t** (*int*) – current time

Raises StopIteration – when the whole sequence is complete

class mindaffectBCI.noisetag.**SingleTrial**(*stimSeq*, *tgtidx: int*, *utopiaController*, *stimulusStateStack*, *numframes: int = None*, *framesperbit: int = 1*, *selectionThreshold: float = None*, *duration: float = 4*, *cueduration: float = 1*, *feedbackduration: float = 1*, *waitduration: float = 1*, *cueframes: int = None*, *feedbackframes: int = None*, *waitframes: int = None*, *permute: bool = False*)

Bases: *mindaffectBCI.noisetag.FSM*

do a complete single trial with: cue->wait->flicker->feedback

next(*t*)

get the next state in the sequence, moving through the single trail stages of cue->wait->flicker->feedback

Parameters **t** (*int*) – current time

Raises StopIteration – when the whole sequence is complete

class mindaffectBCI.noisetag.**WaitFor**(*numframes*)

Bases: *mindaffectBCI.noisetag.FSM*

state machine which waits for given number of frames to pass

get()

get the current state, for this set of active objects

Returns

The current display state information as a 4-tuple with structure: (stimState, target_idx, objIDs, sendEvent) stimState (list-int): the stimulus state for each object as an integer target_idx (int): the index into stimState of the cued target, or -1 if no target set objIDs (list-int): the objectIDs for each of the outputs sendEvent (bool): flag if we should send stimulus events in this state

Return type tuple

next(*t*)

stop after the desired number of frames has passed

Parameters **t** (*int*) – current time stamp

Raises StopIteration – when desired number frames expired

mindaffectBCI.noisetag.**doFrame**(*t*, *stimState*, *tgt_idx=-1*, *objIDs=None*, *utopiaController=None*)

utility function to print a summary of a single frame of a stimulus sequence

Parameters

- **t** (*int*) – current time
- **stimState** (*list-of-int*) – the current stimulus state for each output.
- **tgt_idx** (*int, optional*) – the index of the cued target (if set), -1 if no target. Defaults to -1.
- **objIDs** (*list-of-int, optional*) – the used object ids for each output. Defaults to None.
- **utopiaController** (UtopiaController, *optional*) – the controller for communcation to the decoder. Defaults to None.

mindaffectBCI.noisetag.**mkBlinkingSequence**(*numframes*, *tgtidx*, *tgtState=1*)

make a simple on-off blinking sequence

Parameters

- **numframes** (*int*) – number of frame to blink for
- **tgtidx** (*int*) – the index of the output to blink
- **tgtState** (*int, optional*) – the state to show for the blinking target. Defaults to 1.

Returns (numframes,255) stimulus state for each output at each timepoint

Return type list-of-lists

class mindaffectBCI.noisetag.**sumstats**(*bufsize: int = 120*)
Bases: object

Utility class to record summary stastics for, e.g. frame flip timing

addpoint(*x*)
add a point to the running buffer of statistics

Parameters **x** (*float*) – the point to add

hist()
get the histogram of statistics

Returns string summary of the data histogram

Return type str

update_statistics()
compute updated summary statistics, including: mu=average, med=median, sigma=std-dev, min=min, max=max

mindaffectBCI.online_bci module

class mindaffectBCI.online_bci.**NoneProc**
Bases: object

tempory class simulating a working null sub-process

is_alive()

join()

terminate()

mindaffectBCI.online_bci.**check_is_running**(*hub=None*, *acquisition=None*, *decoder=None*)
check if the background processes are still running

Parameters

- **hub_process** (*[type], optional*) – the hub subprocess. Defaults to hub_process.
- **acquisition_process** (*[type], optional*) – the acquisation subprocess. Defaults to acquisition_process.
- **decoder_process** (*[type], optional*) – the decoder subprocess. Defaults to decoder_process.

Returns true if all are running else false

Return type bool

mindaffectBCI.online_bci.**load_config**(*config_file*)
load an online-bci configuration from a json file

Parameters **config_file** (*[str, file-like]*) – the file to load the configuration from.

mindaffectBCI.online_bci.**parse_args**()
load settings from the json config-file, parse command line arguments, and merge the two sets of settings.

Returns the combined arguments name-space

Return type NameSpace

mindaffectBCI.online_bci.**run**(*label=''*, *logdir=None*, *block=True*, *acquisition=None*, *acq_args=None*, *decoder='decoder'*, *decoder_args=None*, *presentation='selectionMatrix'*, *presentation_args=None*)
[summary]

Parameters

- **label** (*str, optional*) – string label for the saved data file. Defaults to ''.
- **logdir** (*str, optional*) – directory to save log files / data files. Defaults to None = $installdir$/logs.
- **acquisition** (*str, optional*) – the name of the acquisition driver to use. Defaults to None.
- **acq_args** (*dict, optional*) – dictionary of optoins to pass to the acquisition driver. Defaults to None.
- **decoder** (*str, optional*) – the name of the decoder function to use. Defaults to 'decoder'.
- **decoder_args** (*dict, optional*) – dictinoary of options to pass to the mindaffectBCI.decoder.run(). Defaults to None.
- **presentation** (*str, optional*) – the name of the presentation function to use. Defaults to: 'selectionMatrix'
- **presentation_args** (*dict, optional*) – dictionary of options to pass to mindaffectBCI.examples.presentation.selectionMatrix.run(). Defaults to None.
- **block** (*bool, optional*) – return immeadiately or wait for presentation to finish and then terminate all processes. Default to True

Raises ValueError – invalid options, e.g. unrecognised acq_driver

mindaffectBCI.online_bci.**shutdown**(*hub=None*, *acquisition=None*, *decoder=None*)
shutdown any background processes started for the BCI

Parameters

- **hub** (*subprocess, optional*) – handle to the hub subprocess object. Defaults to hub_process.
- **acquisition** (*subprocess, optional*) – the acquisatin subprocess object. Defaults to acquisition_process.
- **decoder** (*subprocess, optional*) – the decoder subprocess object. Defaults to decoder_process.

mindaffectBCI.online_bci.**startDecoderProcess**(*decoder*, *decoder_args*, *label='online_bci'*, *logdir=None*)
start the EEG decoder process

Parameters

- **label** (*str*) – a textual name for this process

- **decoder** (*str*) – the name for the acquisition device to start. One-of: 'decoder' - use the mindaffectBCI.decoder.decoder 'none' - don't start a decoder
- **decoder_args** (*dict*) – dictionary of additional arguments to pass to the decoder
- **logdir** (*str, optional*) – directory to save log/save files.

Raises ValueError – unrecognised arguments, e.g. acquisition type.

Returns sub-process for managing the started decoder

Return type Process

mindaffectBCI.online_bci.**startHubProcess**(*label='online_bci'*, *logdir=None*)

Start the process to manage the central utopia-hub

Parameters **label** (*str*) – a textual name for this process

Raises ValueError – unrecognised arguments, e.g. acquisition type.

Returns sub-process for managing the started acquisition driver

Return type hub (Process)

mindaffectBCI.online_bci.**startPresentationProcess**(*presentation*, *presentation_args*)

mindaffectBCI.online_bci.**startacquisitionProcess**(*acquisition*, *acq_args*, *label='online_bci'*, *logdir=None*)

Start the process to manage the acquisition of data from the amplifier

Parameters

- **label** (*str*) – a textual name for this process
- **acquisition** (*str*) – the name for the acquisition device to start. One-of: 'none' - do nothing, 'brainflow' - use the mindaffectBCI.examples.acquisition.utopia_brainflow driver 'fakedata'- start a fake-data streamer 'eego' - start the ANT-neuro eego driver 'lsl' - start the lsl EEG sync driver
- **acq_args** (*dict*) – dictionary of additional arguments to pass to the acquisition device

Raises ValueError – unrecognised arguments, e.g. acquisition type.

Returns sub-process for managing the started acquisition driver

Return type Process

mindaffectBCI.online_bci_pi module

mindaffectBCI.ssdpDiscover module

mindaffectBCI.ssdpDiscover.**discoverOrIPscan**(*port: int = 8400*, *timeout_ms: int = 15000*)

[summary]

Parameters

- **port** (*int, optional*) – [description]. Defaults to 8400.
- **timeout_ms** (*int, optional*) – [description]. Defaults to 15000.

Returns [description]

Return type [type]

mindaffectBCI.ssdpDiscover.**get_all_ips**()

[summary]

Returns [description]

Return type [type]

mindaffectBCI.ssdpDiscover.**get_ip_address**(*NICname*)
[summary]

Parameters **NICname** (*[type]*) – [description]

Returns [description]

Return type [type]

mindaffectBCI.ssdpDiscover.**get_local_ip**()
[summary]

Returns [description]

Return type [type]

mindaffectBCI.ssdpDiscover.**get_remote_ip**()
[summary]

Returns [description]

Return type [type]

mindaffectBCI.ssdpDiscover.**ip_is_local**(*ip_string*)
Uses a regex to determine if the input ip is on a local network. Returns a boolean. It's safe here, but never use a regex for IP verification if from a potentially dangerous source.

Parameters **ip_string** (*[type]*) – [description]

Returns [description]

Return type [type]

mindaffectBCI.ssdpDiscover.**ipscanDiscover**(*port: int*, *ip: str = None*, *timeout=0.5*)
scan for service by trying all 255 possible final ip-addresses

Parameters

- **port** (*int*) – [description]
- **ip** (*str, optional*) – [description]. Defaults to None.
- **timeout** (*float, optional*) – [description]. Defaults to .5.

Returns [description]

Return type [type]

mindaffectBCI.ssdpDiscover.**nic_info**()
Return a list with tuples containing NIC and IPv4 address

Returns [description]

Return type [type]

class mindaffectBCI.ssdpDiscover.**ssdpDiscover**(*servicetype='ssdp:all'*, *discoverytimeout=3*)
Bases: object

[summary]

Returns [description]

Return type [type]

discover(*timeout=0.001*, *querytimeout=5*, *v6=False*, *intf=None*)

auto-discover the utopia-hub using ssdp discover messages, timeout is time to wait for a response. query timeout is time between re-sending the ssdp-discovery query message.

Parameters

- **timeout** (*float, optional*) – [description]. Defaults to .001.
- **querytimeout** (*int, optional*) – [description]. Defaults to 5.
- **v6** (*bool, optional*) – [description]. Defaults to False.
- **intf** (*[type], optional*) – [description]. Defaults to None.

Returns [description]

Return type [type]

initSocket(*v6=False*, *intf=None*)
[summary]

Parameters

- **v6** (*bool, optional*) – [description]. Defaults to False.
- **intf** (*[type], optional*) – [description]. Defaults to None.

makeDiscoveryMessage(*ssdpgroup*, *servicetype*, *timeout*)
[summary]

Parameters

- **ssdpgroup** (*[type]*) – [description]
- **servicetype** (*[type]*) – [description]
- **timeout** (*[type]*) – [description]

Returns [description]

Return type [type]

msearchTemplate = 'M-SEARCH * HTTP/1.1\r\nHOST: {0}:{1}\r\nMAN: "ssdp:discover"\r\nST:

ssdpgroup = None

ssdpv4group = ('239.255.255.250', 1900)

ssdpv6group = ('FF08::C', 1900)

wait_msearch_response(*timeout*, *verb=0*)
[summary]

Parameters

- **timeout** (*[type]*) – [description]
- **verb** (*int, optional*) – [description]. Defaults to 0.

Returns [description]

Return type [type]

mindaffectBCI.ssdpDiscover.**testIncrementalScanning**()
[summary]

mindaffectBCI.stimseq module

class mindaffectBCI.stimseq.**StimSeq**(*st=None*, *ss=None*, *es=None*)
Bases: object

[summary]

Raises

- Exception – [description]
- Exception – [description]

Returns [description]

Return type [type]

convertstimSeq2int(*scale=1*)
[summary]

Parameters **scale** (*int, optional*) – [description]. Defaults to 1.

eventSeq = None

static float2int(*stimSeq*, *scale=1*, *minval=None*, *maxval=None*)
convert float list of lists to integer

Parameters

- **stimSeq** (*[type]*) – [description]
- **scale** (*int, optional*) – [description]. Defaults to 1.
- **minval** (*[type], optional*) – [description]. Defaults to None.
- **maxval** (*[type], optional*) – [description]. Defaults to None.

Returns [description]

Return type [type]

static fromFile(*fname*)
read a stimulus-sequence from a file on disk

Parameters **fname** (*[type]*) – [description]

Returns [description]

Return type [type]

static fromString(*f*)
read a stimulus-sequence definition from a string

Parameters **f** (*[type]*) – [description]

Raises Exception – [description]

Returns [description]

Return type [type]

static readArray(*f*, *width=-1*)
[summary]

Parameters

- **f** (*[type]*) – [description]
- **width** (*int, optional*) – [description]. Defaults to -1.

Raises `Exception` – [description]

Returns [description]

Return type [type]

setStimRate(*rate*)
rewrite the stimtimes to equal given rate in hz

Parameters **rate** (*[type]*) – [description]

stimSeq = None

stimTime_ms = None

toFile(*fname*)
[summary]

Parameters **fname** (*[type]*) – [description]

mindaffectBCI.stimseq.**mkCodes**()
[summary]

mindaffectBCI.stimseq.**mkFreqTag**(*period_phase=((4, 0), (5, 0), (6, 0), (7, 0), (8, 0), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (3, 2), (4, 2), (5, 2), (6, 2), (7, 2), (8, 2), (4, 3), (5, 3), (6, 3), (7, 3), (8, 3), (5, 4), (6, 4), (7, 4), (8, 4), (6, 5), (7, 5), (8, 5), (7, 6), (8, 6), (8, 7))*, *nEvent=120*, *isbinary=True*)
Generate a frequency tagging stimulus sequence

Parameters

- **period_phase** (*list, optional*) – List of stimulus periods and phase offsets expressed in events. Defaults to all phases from 2-5 long.
- **nEvent** (*int, optional*) – total length of the sequence to generate. (To avoid steps should be LCM of periods) Defaults to 120.
- **isbinary** (*bool, optional*) – flag if we generate a binary sequence or continuous

mindaffectBCI.stimseq.**mkRowCol**(*width=5*, *height=5*, *repeats=10*)
Make a row-column stimulus sequence

Parameters

- **width** (*int, optional*) – width of the matrix. Defaults to 5.
- **height** (*int, optional*) – height of the matrix. Defaults to 5.
- **repeats** (*int, optional*) – number of random row->col repeats. Defaults to 10.

Returns The generated stimulus sequence

Return type [*StimSeq*]

mindaffectBCI.stimseq.**setStimRate**(*stimTime_ms*, *framerate*)
rewrite the stim-times to a new frequency

Parameters

- **stimTime_ms** (*[type]*) – [description]
- **framerate** (*[type]*) – [description]

Returns [description]

Return type [type]

mindaffectBCI.stimseq.**transpose**(*M*)
[summary]

Parameters **M** (*[type]*) – [description]

Returns [description]

Return type [type]

mindaffectBCI.utopia2output module

class mindaffectBCI.utopia2output.**Utopia2Output**(*outputPressThreshold=None*, *outputReleaseThreshold=None*, *objectID2Action=None*)
Bases: object

Example class for a utopia OUTPUT module. Connects to the utopia server and then either, depending on mode:

a) listens for output which exceed it's probability threshold before

then printing them and using NEWTARGET to indicated the output has taken place

connect(*host=None*, *port=None*, *timeout_ms=30000*)
[summary]

Parameters

- **host** (*[type], optional*) – [description]. Defaults to None.
- **port** (*[type], optional*) – [description]. Defaults to None.
- **timeout_ms** (*int, optional*) – [description]. Defaults to 30000.

doOutput(*objID*)
This function is run when objID has sufficiently low error to mean that and output should be generated for this objID. N.B. Override/Replace this function with your specific output method.

perrModeOutput(*msgs*)

process a perr-message generating appropriate output. To avoid 'key-bounce' we use a press-release semantics, where the output is 'Pressed' so the output is generated when the Perr < outputPressThreshold, then further output is inhibited until Perr > outputReleaseThreshold.

Parameters **msgs** (*[type]*) – [description]

run(*timeout_ms=3000*)
mainloop of utopia-to-output mapping runs an infinite loop, waiting for new messages from utopia, filtering out those mesages which contain an output prediction (i.e. PREDICTEDTARGETPROB message) and if the output prediction is sufficiently confident forwarding this to the output device and sending a NEWTARGET to the recogniser to indicate the output was sent

Parameters **timeout_ms** (*int, optional*) – [description]. Defaults to 3000.

selectionModeOutput(*msgs*)
Process selection message to generate output. Basically generate output if the messages objectID is one of the ones we are tasked with generating output for

Parameters **msgs** (*[type]*) – [description]

mindaffectBCI.utopiaController module

class mindaffectBCI.utopiaController.**UtopiaController**(*clientid=None*)
Bases: object

controller class to manage the interaction with the Mindaffect decoder, setting up the connection, sending and recieving messages, and firing message event handlers

Raises ValueError – [description]

Returns [description]

Return type [type]

addMessageHandler(*cb*)
[summary]

Parameters **cb** (*function*) – [description]

addPredictionHandler(*cb*)
[summary]

Parameters **cb** (*function*) – [description]

addSelectionHandler(*cb*)
[summary]

Parameters **cb** (*function*) – [description]

addSignalQualityHandler(*cb*)
[summary]

Parameters **cb** (*function*) – [description]

addSubscription(*msgs*)
[summary]

Parameters **msgs** (*[type]*) – [description]

autoconnect(*host=None*, *port=8400*, *timeout_ms=5000*, *localhostifhostnotfound=True*, *queryifhostnotfound=True*, *scanifhostnotfound=False*)
[summary]

Parameters

- **host** (*[type], optional*) – [description]. Defaults to None.
- **port** (*int, optional*) – [description]. Defaults to 8400.
- **timeout_ms** (*int, optional*) – [description]. Defaults to 5000.
- **localhostifhostnotfound** (*bool, optional*) – [description]. Defaults to True.
- **queryifhostnotfound** (*bool, optional*) – [description]. Defaults to True.
- **scanifhostnotfound** (*bool, optional*) – [description]. Defaults to False.

clearLastPrediction()
clear the last predicted target

getLastPrediction()
check for new predictions from the utopia-decoder

Returns [description]

Return type [type]

getLastSelection()
check if any object prediction is high enough for it to be selected

Returns [description]

Return type [type]

getLastSignalQuality()
[summary]

Returns [description]

Return type [type]

getNewMessages(*timeout_ms=0*)
get new messages from the utopia-hub, and store the list of new

Parameters **timeout_ms** (*int, optional*) – [description]. Defaults to 0.

Returns [description]

Return type [type]

getTimeStamp()
get a (relative) wall-time stamp *in milliseconds*

gethostport()

isConnected()

log(*msg*)
[summary]

Parameters **msg** (*[type]*) – [description]

mkStimulusEvent(*stimulusState*, *timestamp=None*, *targetState=None*, *objIDs=None*)
make a valid stimulus event for the given stimulus state

Parameters

- **stimulusState** (*[type]*) – [description]
- **timestamp** (*[type], optional*) – [description]. Defaults to None.
- **targetState** (*[type], optional*) – [description]. Defaults to None.
- **objIDs** (*[type], optional*) – [description]. Defaults to None.

Raises ValueError – [description]

Returns [description]

Return type [type]

modeChange(*newmode*)
[summary]

Parameters **newmode** (*[type]*) – [description]

newTarget()
[summary]

removeSubscription(*msgs*)
[summary]

Parameters **msgs** (*[type]*) – [description]

selection(*objID*)
[summary]

Parameters **objID** (*[type]*) – [description]

sendStimulusEvent(*stimulusState*, *timestamp=None*, *targetState=None*, *objIDs=None*)
Send a message to the Utopia-HUB informing of the current stimulus state

Parameters

- **stimulusState** (*[type]*) – [description]
- **timestamp** (*[type], optional*) – [description]. Defaults to None.
- **targetState** (*[type], optional*) – [description]. Defaults to None.
- **objIDs** (*[type], optional*) – [description]. Defaults to None.

Returns [description]

Return type [type]

setTimeStampClock(*tsclock*)
[summary]

Parameters **tsclock** (*[type]*) – [description]

Returns [description]

Return type [type]

subscribe(*msgs=None*)
[summary]

Parameters **msgs** (*[type], optional*) – [description]. Defaults to None.

mindaffectBCI.utopiaController.**injectERP**(*amp*, *host='localhost'*, *port=8300*)
Inject an erp into a simulated data-stream, sliently ignore if failed, e.g. because not simulated

Parameters

- **amp** (*[type]*) – [description]
- **host** (*str, optional*) – [description]. Defaults to "localhost".
- **port** (*int, optional*) – [description]. Defaults to 8300.

mindaffectBCI.utopiaController.**newMessageHandler**(*msgs*)
[summary]

Parameters **msgs** (*[type]*) – [description]

mindaffectBCI.utopiaclient module

This module contains the message classes for communication with the mindaffect BCI hub and a simple client for connecting to and managing this connection.

class mindaffectBCI.utopiaclient.**DataHeader**(*timestamp=None*, *fsample=None*, *nchannels=None*, *labels=None*)
Bases: *mindaffectBCI.utopiaclient.UtopiaMessage*

the DATAHEADER utopia message class – which is used to give general meta-information about the EEG stream

static deserialize(*buf*)
Static method to create a HEADER class from a **PAYLOAD** byte-stream, return created object and the number of bytes consumed from buf

Parameters **buf** (*bytes*) – the message buffer

Returns the decoded message

Return type *UtopiaMessage*

msgID = 65

msgName = 'DATAHEADER'

serialize()

Returns the contents of this event as a byte-stream, ready to send over the network, or None in case of conversion problems.

Returns the encoded message

Return type bytes

class mindaffectBCI.utopiaclient.**DataPacket**(*timestamp=None*, *samples=None*)
Bases: *mindaffectBCI.utopiaclient.UtopiaMessage*

the DATAPACKET utopia message class — which is used to stream raw (time,channels) EEG data packets.

static deserialize(*buf*)
Static method to create a STIMULUSEVENT class from a **PAYLOAD** byte-stream, return created object and the number of bytes consumed from buf

Parameters **buf** (*bytes*) – the message buffer

Returns the decoded message

Return type *UtopiaMessage*

msgID = 68

msgName = 'DATAPACKET'

serialize()

Returns the contents of this event as a byte-stream, ready to send over the network, or None in case of conversion problems.

Returns the encoded message

Return type bytes

class mindaffectBCI.utopiaclient.**Heartbeat**(*timestamp: int = None*, *statemessage: str = None*)
Bases: *mindaffectBCI.utopiaclient.UtopiaMessage*

the HEARTBEAT utopia message class

static deserialize(*buf*)
Static method to create a HEARTBEAT class from a **PAYLOAD** byte-stream, return the number of bytes consumed from buf

Parameters **buf** (*bytes*) – the message buffer

Returns the decoded message

Return type *Heartbeat*

msgID = 72

msgName = 'HEARTBEAT'

serialize()
Returns the contents of this event as a string, ready to send over the network, or None in case of conversion problems.

Returns the serialized version of this message

Return type bytes

class mindaffectBCI.utopiaclient.**Log**(*timestamp: int = None*, *logmsg: str = None*)
Bases: *mindaffectBCI.utopiaclient.UtopiaMessage*

the LOG utopia message class – which is used to send arbitary log messages to the hub

static deserialize(*buf*)
Static method to create a MODECHANGE class from a **PAYLOAD** byte-stream, return the number of bytes consumed from buf

Parameters **buf** (*bytes*) – the message buffer

Returns the decoded message

Return type *UtopiaMessage*

msgID = 76

msgName = 'LOG'

serialize()

Returns the contents of this event as a byte-stream, ready to send over the network, or None in case of conversion problems.

Returns the encoded message

Return type bytes

class mindaffectBCI.utopiaclient.**ModeChange**(*timestamp=None*, *newmode=None*)
Bases: *mindaffectBCI.utopiaclient.UtopiaMessage*

the MODECHANGE utopia message class – which is used to tell the decoder to change system modes, for example to switch to calibration-mode.

static deserialize(*buf*)
Static method to create a MODECHANGE class from a **PAYLOAD** byte-stream, return the number of bytes consumed from buf

Parameters **buf** (*bytes*) – the message buffer

Returns the decoded message

Return type *UtopiaMessage*

msgID = 77

msgName = 'MODECHANGE'

serialize()

Returns the contents of this event as a byte-stream, ready to send over the network, or None in case of conversion problems.

Returns the encoded message

Return type bytes

class mindaffectBCI.utopiaclient.**NewTarget**(*timestamp=None*)
Bases: *mindaffectBCI.utopiaclient.UtopiaMessage*

the NEWTARGET utopia message class – which is used to tell other clients that a new target selection round has begun.

static deserialize(*buf*)
Static method to create a HEARTBEAT class from a **PAYLOAD** byte-stream, return the number of bytes consumed from buf

Parameters **buf** (*bytes*) – the message buffer

Returns the decoded message

Return type *UtopiaMessage*

msgID = 78

msgName = 'NEWTARGET'

serialize()

Returns the contents of this event as a byte-stream, ready to send over the network, or None in case of conversion problems.

Returns the encoded message

Return type bytes

class mindaffectBCI.utopiaclient.**PredictedTargetDist**(*timestamp=None*, *objIDs=None*, *pTgt=None*)
Bases: *mindaffectBCI.utopiaclient.UtopiaMessage*

the PredictedTargetDist utopia message class – which is used to send information about the current predicted target probabilites for the outputs

static deserialize(*buf*)
Static method to create a PREDICTEDTARGETDIST class from a **PAYLOAD** byte-stream, return created object and the number of bytes consumed from buf

Parameters **buf** (*bytes*) – the message buffer

Returns the decoded message

Return type *UtopiaMessage*

msgID = 70

msgName = 'PREDICTEDTARGETDIST'

serialize()

Returns the contents of this event as a byte-stream, ready to send over the network, or None in case of conversion problems.

Returns the encoded message

Return type bytes

class mindaffectBCI.utopiaclient.**PredictedTargetProb**(*timestamp=None*, *Yest: int = -1*, *Perr: float = 1.0*)
Bases: *mindaffectBCI.utopiaclient.UtopiaMessage*

the PREDICTEDTARGETPROB utopia message class – which is used to communicate the most likely target and the estimate error rate for this prediction.

static deserialize(*buf*)
Static method to create a MODECHANGE class from a **PAYLOAD** byte-stream, return the number of bytes consumed from buf

Parameters **buf** (*bytes*) – the message buffer

Returns the decoded message

Return type *UtopiaMessage*

msgID = 80

msgName = 'PREDICTEDTARGETPROB'

serialize()

Returns the contents of this event as a byte-stream, ready to send over the network, or None in case of conversion problems.

Returns the encoded message

Return type bytes

class mindaffectBCI.utopiaclient.**RawMessage**(*msgID*, *version*, *payload*)
Bases: *mindaffectBCI.utopiaclient.UtopiaMessage*

Class for a raw utopia message, i.e. decoded header but raw payload

classmethod deserialize(*buf*)
Read a raw message from a byte-buffer, return the read message and the number of bytes used from the bytebuffer, or None, 0 if message is mal-formed

Parameters **buf** (*bytes*) – the byte buffer to read from

Returns the decoded raw message

Return type *RawMessage*

classmethod deserializeMany(*buf*)
decode multiple RawMessages from the byte-buffer of data, return the length of data consumed.

Parameters **buf** (*bytes*) – the byte buffer to read from

Returns the list of decoded raw messages

Return type list-of-RawMessage

classmethod fromUtopiaMessage(*msg: mindaffectBCI.utopiaclient.UtopiaMessage*)
Construct a raw-message wrapper from a normal payload message

Parameters **msg** (UtopiaMessage) – the message to make to raw

Returns the raw version of this utopia message

Return type *RawMessage*

msgName = 'RAW'

serialize()
convert the raw message to the string to make it network ready

Returns the byte serialized version of this raw message

Return type bytes

class mindaffectBCI.utopiaclient.**Reset**(*timestamp=None*)
Bases: *mindaffectBCI.utopiaclient.UtopiaMessage*

the RESET utopia message class – which is used to tell the decoder to reset and clear it's model information.

static deserialize(*buf*)
Static method to create a RESET class from a **PAYLOAD** byte-stream, return the number of bytes consumed from buf

Parameters **buf** (*bytes*) – the message buffer

Returns the decoded message

Return type *UtopiaMessage*

msgID = 82

msgName = 'RESET'

serialize()

Returns the contents of this event as a byte-stream, ready to send over the network, or None in case of conversion problems.

Returns the encoded message

Return type bytes

class mindaffectBCI.utopiaclient.**Selection**(*timestamp=None*, *objID=None*)
Bases: *mindaffectBCI.utopiaclient.UtopiaMessage*

the SELECTION utopia message class – which is used to inform other clients that a particular output has been selected by **this** client.

static deserialize(*buf*)
Static method to create a SELECTION class from a **PAYLOAD** byte-stream, return the number of bytes consumed from buf

Parameters **buf** (*bytes*) – the message buffer

Returns the decoded message

Return type *UtopiaMessage*

msgID = 83

msgName = 'SELECTION'

serialize()

Returns the contents of this event as a byte-stream, ready to send over the network, or None in case of conversion problems.

Returns the encoded message

Return type bytes

class mindaffectBCI.utopiaclient.**SignalQuality**(*timestamp: int = None*, *signalQuality=None*)
Bases: *mindaffectBCI.utopiaclient.UtopiaMessage*

the SIGNALQUALITY utopia message class – which is used to send information about the estimated signal to noise for each channel in the data stream

static deserialize(*buf*)
Static method to create a SIGNALQUALIYT class from a **PAYLOAD** byte-stream, return the number of bytes consumed from buf

Parameters **buf** (*bytes*) – the message buffer

Returns the decoded message

Return type *UtopiaMessage*

msgID = 81

msgName = 'SIGNALQUALITY'

serialize()

Returns the contents of this event as a byte-stream, ready to send over the network, or None in case of conversion problems.

Returns the encoded message

Return type bytes

class mindaffectBCI.utopiaclient.**StimulusEvent**(*timestamp=None*, *objIDs=None*, *objState=None*)
Bases: *mindaffectBCI.utopiaclient.UtopiaMessage*

the STIMULUSEVENT utopia message class – which is used to send information about the current stimulus state of this client to other clients

static deserialize(*buf*)
Static method to create a STIMULUSEVENT class from a **PAYLOAD** byte-stream, return created object and the number of bytes consumed from buf

Parameters **buf** (*bytes*) – the message buffer

Returns the decoded message

Return type *UtopiaMessage*

msgID = 69

msgName = 'STIMULUSEVENT'

serialize()
Converts this message to a string representation to send over the network

Returns the encoded message

Return type bytes

class mindaffectBCI.utopiaclient.**Subscribe**(*timestamp=None*, *messageIDs=None*)
Bases: *mindaffectBCI.utopiaclient.UtopiaMessage*

the SUBSCRIBE utopia message class – which is used to tell the hub which messages IDs to forward to this client.

static deserialize(*buf*)
Static method to create a SIGNALQUALITY class from a **PAYLOAD** byte-stream, return the number of bytes consumed from buf

Parameters **buf** (*bytes*) – the message buffer

Returns the decoded message

Return type *UtopiaMessage*

msgID = 66

msgName = 'SUBSCRIBE'

serialize()

Returns the contents of this event as a byte-stream, ready to send over the network, or None in case of conversion problems.

Returns the encoded message

Return type bytes

class mindaffectBCI.utopiaclient.**TimeStampClock**
Bases: object

Base class for time-stamp sources. Match this prototype to replace the default timestamp source

getTimeStamp()
get the time-stamp in milliseconds ! N.B. **must** fit in an int32!

Returns the timestamp

Return type int

class mindaffectBCI.utopiaclient.**UtopiaClient**(*clientstate=None*)
Bases: object

Class for managing a client connection to a UtopiaServer.

DEFAULTHOST = 'localhost'

DEFAULTPORT = 8400

HEARTBEATINTERVALUDP_ms = 200

HEARTBEATINTERVAL_ms = 1000

MAXMESSAGESIZE = 1048576

UTOPIA_SSDP_SERVICE = 'utopia/1.1'

autoconnect(*hostname=None*, *port=None*, *timeout_ms=5000*, *queryifhostnotfound=False*, *localhostifhostnotfound=True*, *scanifhostnotfound=False*)
connect to the hub/decoder, auto-discover if explict IP address is not given

Parameters

- **hostname** (*str, optional*) – hostname or IP where the hub resides. Defaults to None.
- **port** (*int, optional*) – port where the host is listening. Defaults to 8400.
- **timeout_ms** (*int, optional*) – timeout in milliseconds for discovery. Defaults to 5000.
- **queryifhostnotfound** (*bool, optional*) – query the user for autodiscover fails. Defaults to False.
- **localhostifhostnotfound** (*bool, optional*) – try localhost if autodiscovery fails. Defaults to True.
- **scanifhostnotfound** (*bool, optional*) – scan all local IP addresses if autodiscovery fails. Defaults to False.

Raises socket.error – if there is a socket error ;)

connect (*hostname=None*, *port=8400*)
connect([hostname, port]) – make a connection, default host:port is localhost:1972

Parameters

- **hostname** (*str, optional*) – hostname or IP where the hub resides. Defaults to None.
- **port** (*int, optional*) – port where the host is listening. Defaults to 8400.

Returns the current connection status

Return type bool

disableHeartbeats ()
‘ stop sending hearbeat messages. Use, e.g. when you want to use your own time-stamp clock.

disconnect ()
disconnect() – close a connection.

enableHeartbeats ()
start sending heartbeat messages every few seconds

getNewMessages (*timeout_ms=250*)
Wait for new messages from the utopia-server (with optional timeout), decode them and return the list of new messages

Parameters **timeout_ms** (*int, optional*) – timeout in milliseconds to wait for the new messages. Defaults to 250.

Returns the list of recieved messages

Return type list-of-UtopiaMessage

getTimeStamp ()
Get the time-stamp for the current time

Returns the current timestamp

Return type int

gethostport ()
[summary]

Returns [description]

Return type [type]

initClockAlign (*delays_ms=[50, 50, 50, 50, 50, 50, 50, 50, 50, 50]*)
Send some initial heartbeat messages to seed the alignment of the server clock with our local clock

Parameters **delays_ms** (*list-of-int, optional*) – delay in milliseconds between sending the heartbeat messages. Defaults to [50]*10.

messagePingPong (*timeout_ms=500*)
Testing system sending 20 messages and printing any responses

Parameters **timeout_ms** (*int, optional*) – delay when waiting for messages. Defaults to 1000.

messagelogger (*timeout_ms=1000*)
Simple message logger, infinite loop waiting for and printing messages from the server

Parameters **timeout_ms** (*int, optional*) – delay when waiting for messages. Defaults to 1000.

recvall (*timeout_ms=0*)
Read all the data from the socket immeaditely or block for timeout_ms if nothing to do.

Parameters **timeout_ms** (*int, optional*) – timeout in milliseconds for the read. Defaults to 0.

Returns the raw stream of bytes from the socket

Return type bytes

sendHeartbeatIfTimeout (*timestamp=None*)
send a heartbeat message if the timeout since the last such message has passed.

Parameters **timestamp** (*int, optional*) – The current timestamp. Defaults to None.

sendMessage (*msg: mindaffectBCI.utopiaclient.UtopiaMessage*)
send a UtopiaMessage to the hub

Parameters **msg** (UtopiaMessage) – the message to send

sendMessages (*msgs*)
sends single or multiple utopia-messages to the utopia server

Parameters **msgs** (*list-of-UtopiaMessage*) – list of messages to send

sendRaw (*request*)
Send all bytes of the string 'request' out to the connected hub/decoder

Parameters **request** (*bytes*) – the raw stream to send

Raises IOError – if the send fails

sendRawUDP (*request*)
send a raw byte stream over a UDP socket

Parameters **request** (*bytes*) – the raw stream to send

setTimeStampClock (*tsClock*)
set the clock to use for timestamping outgoing messages

Parameters **tsClock** (TimeStampClock) – the timestamp clock to use

Raises ValueError – if the tsClock does not have a getTimeStamp method

try_connect (*hostname*, *port=None*, *timeout_ms=5000*)
try to connect to the hub

Parameters

- **hostname** (*str, optional*) – hostname or IP where the hub resides. Defaults to None.
- **port** (*int, optional*) – port where the host is listening. Defaults to 8400.
- **timeout_ms** (*int, optional*) – timeout in milliseconds for discovery. Defaults to 5000.

class mindaffectBCI.utopiaclient.**UtopiaMessage** (*msgID=0*, *msgName=None*, *version=0*)
Bases: object

Class for a generic UtopiaMessage, i.e. the common structure of all messages

mindaffectBCI.utopiaclient.**decodeRawMessage** (*msg*)
decode utopia RawMessages into actual message objects

Parameters **msg** (RawMessage) – the raw message whose payload should be decoded

Returns the decoded specific message type

Return type StimulusEvent|Heartbeat|....

mindaffectBCI.utopiaclient.**decodeRawMessages**(*msgs*)
decode utopia RawMessages into actual message objects

Parameters **msgs** (*list-of-RawMessage*) – list of RawMessages to decode the payload of

Returns list of the fully decoded messages

Return type list-of-messages

mindaffectBCI.utopiaclient.**ssdpDiscover**(*servicetype=None*, *timeout=3*, *numretries=1*)
auto-discover the utopia-hub using ssdp discover messages

Parameters

- **servicetype** (*str, optional*) – the SSDP service type to search for. Defaults to None.
- **timeout** (*int, optional*) – timeout in seconds for the discovery. Defaults to 3.
- **numretries** (*int, optional*) – number of times to retry discovery. Defaults to 1.

Returns list of the IP addresses of the discovered servers

Return type list-of-str

mindaffectBCI.utopiaclient.**testSending**()
test object sending by connecting and sending a set of test messages to the server

mindaffectBCI.utopiaclient.**testSerialization**()
test object serialization and deserialization by encoded and decoding messages

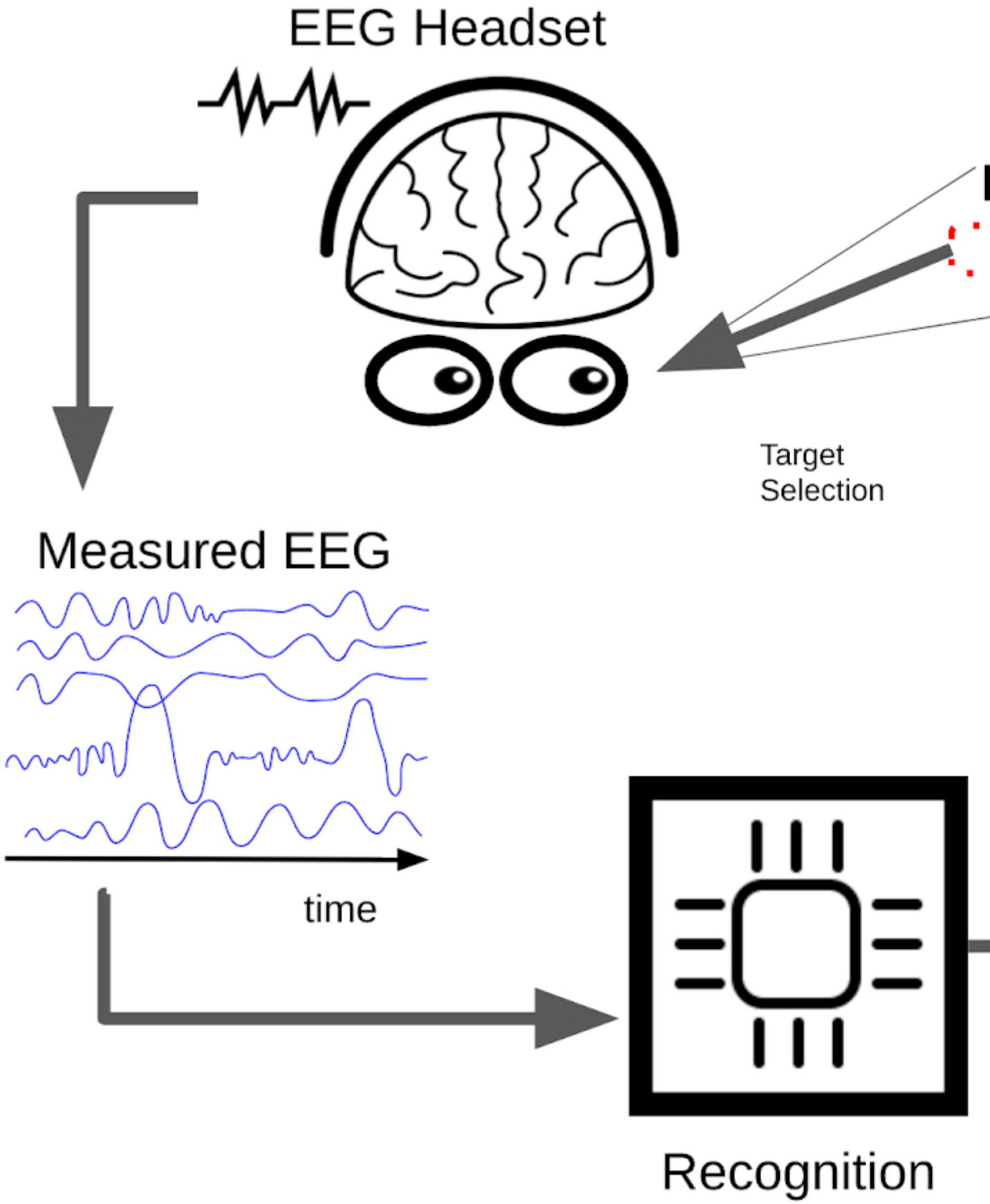
Module contents

3.12 mindaffectBCI : System Overview and Component Roles

3.13 System Overview

The mindaffect BCI is a Brain Computer Interface (BCI) system which allows users to control computers and other electronic devices by looking at flickering objects. As each object has a unique flicker sequence, by measuring the brain's response to this unique sequence the BCI can identify the target object the user is looking at and hence cause that object to be selected.

In the BCI literature this type of BCI is called a visual evoked-response BCI, as the brain response (which we use) is generated (or evoked) by the visual flickering of the target object. Below is an illustration of how a classic evoked response BCI works schematically:


EEG Headset
Target
Selection
Measured EEG
time
Recognition

Evoked Response BCI

To briefly describe this schematic:

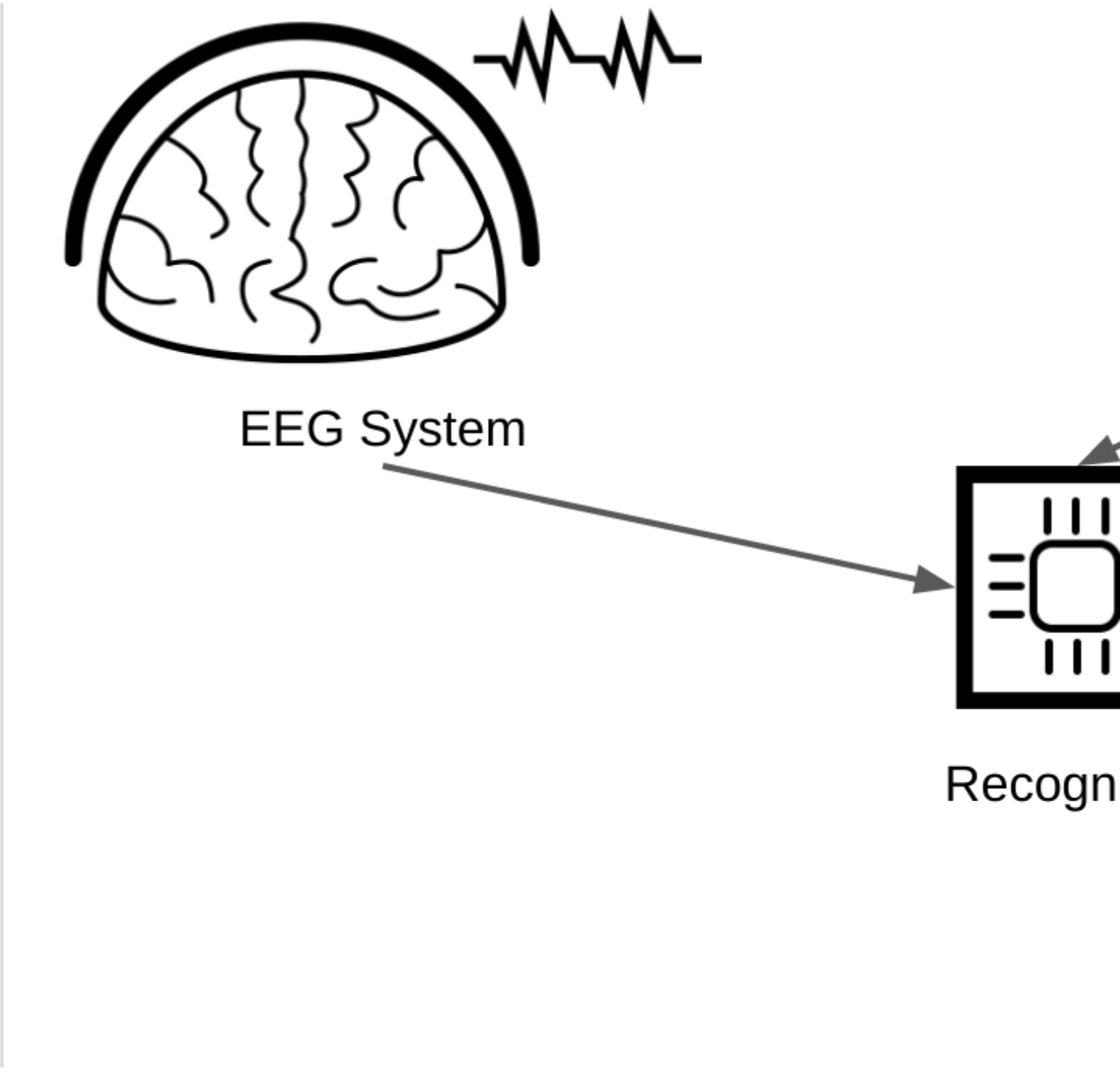
1. Presentation: displays a set of options to the user, such as pickup, drop, and stop.
2. Each of these options then flickers with a given unique flicker sequence (or stimulus-sequence) with different objects getting bright and dark at given times.
3. The user, looks at the option they want to select to select that option.
4. The users brain response to the presented stimulus is measured by EEG - due to the users focus on their target object, this EEG signal contains information on the flicker-sequence of the target object.
5. The recognition system uses the measured EEG and the known stimulus sequence to generate predictions for the probability of the different options being the users target option.
6. Finally, the selection and output system decides when the prediction is confident enough to select an option and generate the desired output – dropping something in this case.

There are many types of (visual) evoked response BCI, all of which work in basically the same fashion. What may differ is the modality in which the stimulus is presented (such as visually, orally, or tactically) and/or the particular properties of the 'flicker' pattern and associated brain response.

The Utopia system uses a particular 'flicker' pattern based on pseudo-random-noise codes, specifically gold-codes, which cause the brain to generate a Visual Evoked Potential. In the BCI literature this type of BCI is called a code-based Visual Evoked Potential (c-VEP).

3.13.1 Physical components of the mindaffect BCI

The main physical components of the mindaffect BCI are illustrated below:



main physical components of the BCI

The 4 main components and their purpose are:

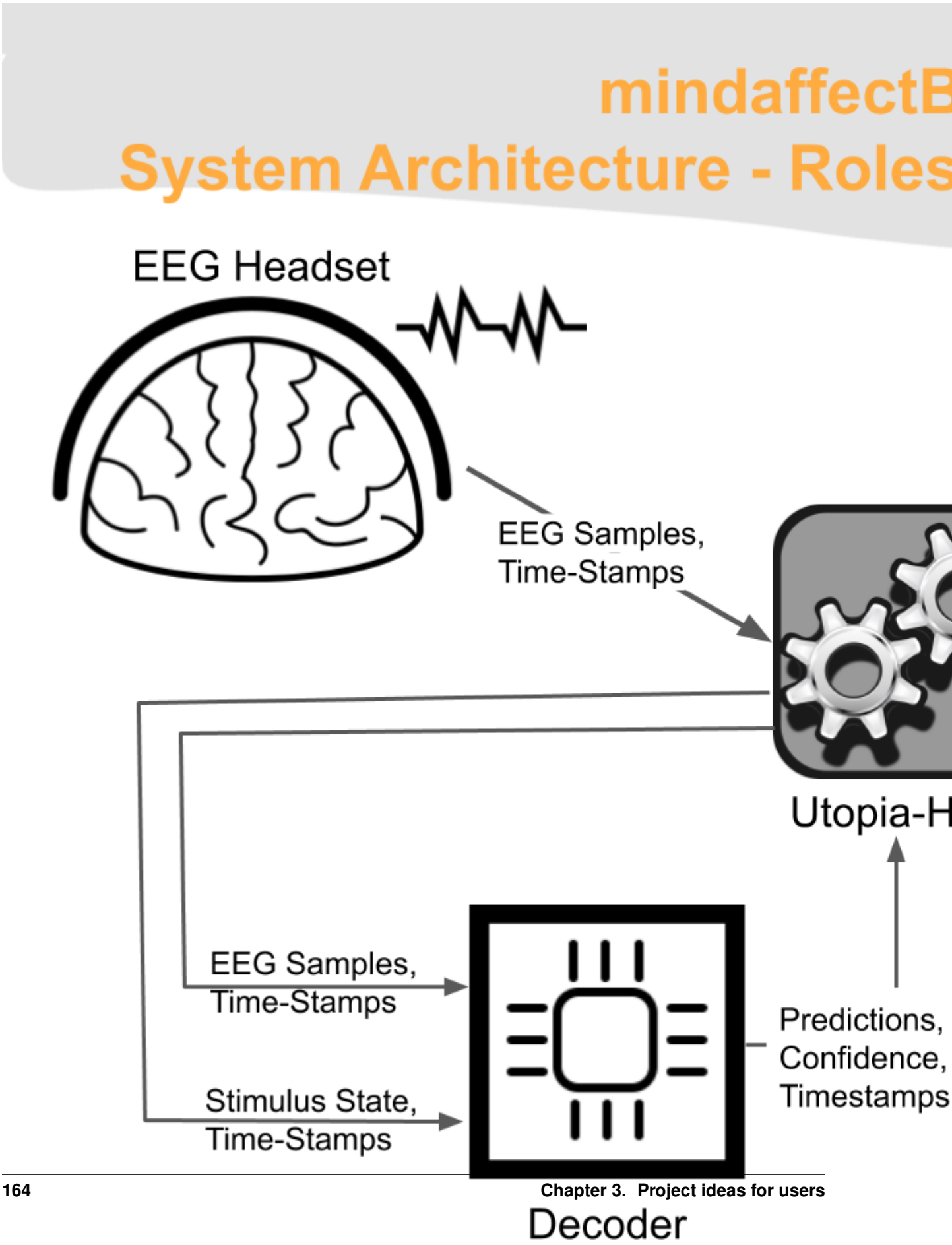
- **Presentation**: This component is usually a screen of some sort, e.g. LCD, or tablet-computer, but may be any system which is able to generate a rapid controlled visual change, such as an LED array or a laser-pointer. The purpose of this component is to present the selectable objects and 'flicker' them in the required sequence such that when the user looks at their *target object* (that is the object they want to select) a detectable brain response is evoked. It must also send information on the actual flicker sequence of each object to the Reogniser so it can identify the users target object.
- **EEG system:** This component usually consists of 3 sub-components: a) a headset that sits on the users head and

ensures a physical connection between the measurement electrodes and the users scalp. b) an amplifier which measures the relevant brain response and converts it into digital signals, c) a transmission system (such as USB, WiFI or Bluetooth) which can send these measurements to the Decoder so it can identify the users target object.

- **Recogniser** or **Decoder** : This component is the physical host of the software that runs the machine learning algorithms and combines the information on the presented stimulus (from the Presentation component) and the measured brain response (from the EEG component) to **recognise** the users intended target and send these **object selections** to the output component. Usually, this component is physically a single-board computer, such as a raspberry pi, but may be a purely software component running on the CPU of another component (such as CPU of a tablet or desktop computer used for presentation).
- **Output**: In many cases this component is integrated as part of the presentation component, such as a tablet-computer, but may be a completely independent device, such as a robot or TV, or home-automation device (e.g. door opener/closer.) The purpose of this device is to take the **object selections** generated by the Decoder and generate the desired output. For example, in a virtual-keyboard application this may be to add the selected letter to the current sentence, or in a home-automation setting this may be to change to the selected TV channel or open the indicated door.

3.13.2 Functional roles of the Utopia BCI

The physical components of the Utopia BCI map fairly straightforwardly onto the functional roles within the utopia system. The below diagram illustrates the main functional roles of the Utopia system and the communication messages used to communicate between them. The arrows indicate the direction of information flow between components.


mindaffectB
System Architecture - Roles
EEG Headset
EEG Samples,
Time-Stamps
Utopia-H
EEG Samples,
Time-Stamps
Stimulus State,
Time-Stamps
Predictions,
Confidence,
Timestamps
Decoder

main functional roles in the mindaffectBCI

The main functional roles in the utopia system are:

- **Presentation:** The presentation component is responsible for presenting the stimuli to the user, and sending information about this presentation to the **Decoder** so it can identify the users selected target. How the presentation is actually performed is up to the presentation component. In particular the stimulus modality (visual, audio, tactile) and flicker sequence used are the responsibility of the presentation system.

  The stimulus information is sent to the decoder using **Utopia Message-spec** messages (see the Utopia Message-spec for details) using **STIMULUSEVENT** messages. The particular transport used for the messages will depend on the hardware implementation, but currently we use TCP-IP sockets over WIFI. For ease of implementation we provide example presentation implementations with both cVEP stimulus display and TCP-IP based message sending in the following development languages/frameworks on our github page :

  - SWIFT/iOS
  - Unity
  - Python, with pygame
  - Java, with libGDX
  - MATLAB/OCTAVE

- **EEG Acquisition:** The EEG acquisition system is responsible for getting digital EEG samples from the EEG hardware and transmitting them to the **Decoder**.

  This information is transmitted to the decoder in a number of different formats, for example via. Lab Streaming Layer. The mindaffect BCI primarly uses brainflow to interact with the acquisation hardware, and then uses a network-transparent message-specification, see , to forward this data to the **Utopia-Hub**.

- **Utopia-Hub**: This component is responsible for connecting together all the other components of the system. Basically, this is a broker for Utopia Message Spec messages, which accepts incoming messages and forwards them between different clients, for example forwarding all STIMULUSEVENT messages from Presentation clients to the Decoder component and in turn forwarding all PREDICTEDTARGETPROB messages from the Decoder to all output or selection clients.

  The Utopia-Hub is a purely software component running on the same platform as the Decoder. The particular transport used for the messages depends on hardware, but the current system uses TCP-IP sockets over WIFI or Ethernet. In particular the Utopia-Hub allows incoming and outgoing client connections on **TCP port 8400** and **incoming only** connections on **UDP port 8400**.

  The utopia-hub also includes additional functionality making it easier to use and debug the utopia-system including;

  - Sending and monitoring **HEARTBEAT** messages from and to clients. These messages are sent a regular intervals to check for liveness of the clients within the utopia-hub and to detect hub crashes within clients.
  - Providing auto-discovery services, via Simple Service Discovery (SSDP) to allow automatic detection of the utopia-hub from clients.

  As the Utopia-Hub is an internal component to the Decoder component, it's actual operation is not discussed further here. (Though FYI: it is a single-threaded network server implemented in JAVA).

  For ease of implementation in clients we provide a full implementation of utopia-hub **CLIENTS** including auto-discovery, heartbeat monitoring, and message sending and receiving for all message types in the following platforms/frameworks (see XXX for more information):

```
*   SWIFT / iOS
*   C# / Unity
*   Python
```

(continues on next page)

(continued from previous page)

```
*   Java
*   MATLAB/OCTAVE
```

- **Decoder:** This component is responsible for using information on which flicker sequences have been presented to the user (received as **STIMULUSEVENT** messages from the presentation component(s)) and what EEG measurements have been made (received from the EEG Acquisition component) to generate **target predictions** sent as **PREDICTEDTARGETPROB** messages to the selection component. As the predictions of the decoder are uncertain, the target predictions consist of a:
  - Predicted target object - which is the object the decoder thinks is most likely to be the users intended target object.
  - Predicted target error - which is the decoders estimate of the chance that it's identification of the target object is **incorrect**.

  Internally, it does this by using **machine learning** techniques to learn a mapping from EEG to a predicted flicker sequence. This predicted flicker sequence is then compared with information on the actual flicker sequences shown to the user by the presentation component to identify the most likely target object. As the actual techniques used to learn this mapping and compare the predicted and actual flicker sequences are internal to the Decoder component these will not be discussed further here.
- **Selection:** The selection component is responsible for taking **target predictions** as generated by the **Decoder** and turning these into **Selections** for which the output system should generate the desired output. How the target prediction information from the decoder and any contextual information available to the selector is used to make these selections is up to the selection system, for example in a virtual keyboard application the context within a word may be used to make selection of the most likely next letter easier. Note: in many cases, the selection system integrated with that of the output system as this has the contextual information available to improve selections – such as knowing the word typed so far in a virtual keyboard.

  In most cases the selection system is simply one of thresholding on the Predicted target error of the target predictions given by the **Decoder**.

  Information on target predictions is transmitted to the output component using **Utopia Message-spec PREDICTEDTARGETPROB** messages (see the Utopia Message-spec for details). The particular transport used for the messages will depend on the hardware implementation, but currently we use TCP-IP sockets over WIFI.
- **Output**: The output system is responsible for transforming selections made by the user (received as **SELECTION** messages) into their desired output. What this output is and how it is delivered is the responsibility of the output system.

  Note: in many cases, such as with a virtual-keyboard, the output role is integrated with the presentation role.

  Information on user selections is transmitted to the output component using **Utopia Message-spec SELECTION** messages (see the Utopia Message-spec for details). The particular transport used for the messages will depend on the hardware implementation, but currently we use TCP-IP sockets over WIFI.

3.13.3 System Mode Switching: Calibration and Prediction

The core of the Utopia system is the machine learning system in the decoder which transforms EEG + StimulusEvents into target predictions. For this transformation to work the decoder needs so-called **calibration** data from which the ML algorithms are **trained**. To get this calibration data the system works in one of two main modes, Calibration and Prediction.

- **Calibration Mode:** In this mode the system is gathering data with which to train the machine learning module. This means that in this mode the Selection and Output roles are not used. Further in this mode the Presentation module must somehow **instruct the user** which target they should attend to.

- **Prediction Mode:** In this mode the machine learning module has been trained (model fitting has been completed) and the Utopia system can be used to make output selections based on brain responses. This means that in this mode all the components, Recoginser, Presentation, Selection and Output work as described above.

Switching between these two modes is achieved by sending MODECHANGE messages to the utopia-hub (which in turn forwards these messages to all other components.)

3.14 mindaffectBCI : Message Specification

3.14.1 Purpose

This document describes in the messages which are passed between the different components of the mindaffect BCI system, and the low-level byte-struture used for the messages. The message specification is *transport agnostic* in that the messages themselves may be sent over different 'wire-protocols', such as BTLE, UDP, TCP, etc.

Note: This specification is intended for developing new low-level components to interface to the mindaffectBCI. Most developers / users can directly use one of the provided higher-level APIs, such as python, java, c#/unity.

3.14.2 Objectives

- Stateless: as much as possible messages are self-contained information updates such that the interpretation of a message depends minimally on the reception of other messages. In this way, we are both robust to loss of a single message and failure/rebooting of an individual component, and also simplify the later loading/replay of saved data as we can basically start playback at any point. The disadvantage of this is however the messages tend to be longer than needed due to the additional redundancy.
- Simple + Readable: as much as possible the message spec will be human readable, such that for example message types are encoded in plain strings. However, the spec will also be simple to read manipulate the messages, thus integers will be sent as integers, and arrays as packed binary arrays.
- Efficient: as much as possible (without violating the stateless and readable objectives) the message spec will be efficient in space usage in transmission.
- Compact : some of our transport layers have very small payload sizes (e.g. BLE has 20 bytes) thus the messages should be compact such that a useful stateless message can be sent in this payload size.
- Latency tolerant : we cannot guarantee timely transmission of messages between components. Thus, the spec will (where appropriate) include additional time-stamp information to allow a 'true' message time-line to be reconstructed.

3.14.3 Structure

The message specification is structured as follows:

- Message Name: a human readable informative name for the message
- Message UID : ascii character used to uniquely identify this message. This is required to be the first character of any message.
- Sender: the component which produces the message
- Receiver: the component which receives the message
- Purpose: the reason for sending this message between these two components

- Format: the detailed structure of the message in terms of the basic types (i.e. char, int8, int16, uint8, single, double etc.) it is made up from. Each slot of the format has:
  - Name:
  - Type Information:
  - Comment: human readable description of the purpose of this slot

3.14.4 Message Specifications

3.14.5 Endianness

To simplify things we *require* that all numbers be encoded in **LITTLE ENDIAN**.

3.14.6 Message Header

All messages start with a standard header consisting of:

3.14.7 General Utility Messages

3.14.8 Presentation -> Recogniser

**STIMULUSDICT **structure

NOTES:

- **objectID **- an object UID is (as the name implies) a ****unique** **identifier for a particular stimulus object. This is used ****both** **for indication of the stimulus state of an object **and **for indication of the identified target object when predictions are generated.

  Object IDs for which *no* stimulus state information has been provided since the last decoder reset command are assumed to not be stimulated. Object IDs do *not* have to be consecutive, and can be allocated arbitarly by the STIMULUS component

- **Stimulusstate **- The **encoding** used by the stimulus state is flexible in this version of the spec. By convention stimulus state is treated as a **single **continuous level of the stimulus intensity, e.g. a grey-scale value. However other encoding formats as possible without changing the message spec. Example encodings could be; bit 0=long, bit(1)=short, or bits 0-4 for intensity and bits5-8 for color. **It is the responsibility of CONFIG to ensure that STIMULUS and RECOGNISER agree on the interperation of the stimulus state object. **

- Example Bandwidth Requirements: for a 36 output display with messages packed into 20bytes (as in BLE). We have 6 bytes header overhead, leaving 7 objects in the message packet. Thus we require 6 packets for a update on all objects in the display, i.e. 6*20 = 120 bytes / display update. Thus @ 60 display rate we require: 120*60 = 7200 bytes/sec. The spec for BLE gives, an *application* data rate of .27Mbit/sec = 270000 bit/sec = 33750 byte/sec. Thus we need about 25% of the total available BLE bandwidth for this common use-case.

3.14.9 Recogniser -> Selection or Output

TARGETERRORDIST

3.14.10 Controller or Output -> All

3.14.11 Recogniser -> User Interface

3.14.12 Acquisition -> Recogniser

Notes:

32bit timestamps @1ms accuracy means the timestamps will wrap-around in 4294967296/1000/60/60/24 = 50 days.. Which is way more than we really need....

With 24 bits this would be 4hr.. For implementation simplicity standard 32bit ints are prefered.

3.14.13 Extension Messages

In this section are listed messages which may be useful in future, but will not be implemented in the V1.0 version of the system.

3.14.14 Config -> Recogniser

3.14.15 Other

3.14.16 Bluetooth Low Energy

When using BLE to communicate we need to define Services and Charactersitics. Services represent groups of functionality which is exposed to other devices, with Characteristics the values to be communicated. Within this message specification a unique characteristic maps directly onto a message type, with groups of messages together making a logically connected service. Specification. Based on this reasoning we have the following BLE specification:

3.14.17 Service: Presentation

UUID: **d3560000-b9ff-11ea-b3de-0242ac130004

Description:

Provides the generic interface for devices which can take on the presentation role, that is which show stimuli to the user for which the brain response will be used to make selections later.

Characteristic: Stimulus State:

UUID: **d3560001-b9ff-11ea-b3de-0242ac130004

Description:

Provides the characteristic to communicate the current device stimulus state to the decoder by formatting STIMULUSEVENT messages.

Properties:

Variable length

Read, Write, Notify.

Payload Format:

Read, Notify: encode the current stimulus state in a STIMULUSEVENT message – Note: this is a *full* message with type, version, length encoding (Later we may remove this technically unnecessary header information), thus it may

take *more than* 1 message to communicate a full stimulus state when we have > 8 objectIDs on a single device. Note however, that as stimulus state's are 'latched' only changed objects need to have their state communicated which may be used to reduce the communication load in resource constrained situations.

(Note: be sure to use the onCharacteristicWrite callback to stream these multi-packet writes in a good way!)

TODO: V2 StimulusEvent spec with binary stimulus state encoding.

Write: encode control messages for the presentation device. Basically this includes:

NEWTARGET messages

SELECTION messages.

Again, we encode the full message spec in the BLE characteristic write (including header, version, length).

3.14.18 Service: Decoder

UUID: **d3560100-b9ff-11ea-b3de-0242ac130004

Description:

Provides the general interface for the EEG decoder. Thus, it consumes STIMULUSEVENT messages and generates Prediction messages.

Characteristic: PredictedTargetProb:

UUID: **d3560101-b9ff-11ea-b3de-0242ac130004

Description:

Provides the characteristic to communicate the predicted target probability by formatting PREDICTEDTARGET-PROB messages.

Properties:

Read, Notify.

Payload Format:

Read, Notify: encode the current predicted target prob as a PREDICTEDTARGETPROB message. Note this is the full message including headers.

Characteristic: PredictedTargetDist:

UUID: **d3560102-b9ff-11ea-b3de-0242ac13000423c

Description:

Provides the characteristic to communicate the predicted target probability by formatting PREDICTEDTARGETDIST messages.

Properties:

Read, Notify.

Payload Format:

Read, Notify: encode the current predicted target prob as a PREDICTEDTARGETDIST message. Note: this is a *full* message with type, version, length encoding (Later we may remove this technically unnecessary header information), thus it may take *more than* 1 message to communicate a full prediction state when we have > 8 objectIDs on a single device. As all messages are both time-stamped and stateless, this will be achieved by simply cutting the messages into smaller pieces to be transmitted one after each other.

3.14.19 Service: Selection

UUID: **d3560200-b9ff-11ea-b3de-0242ac130004

Description:

Provide the ability to receive and act on selections by the BCI.

Characteristic: Selection:

UUID: **d3560201-b9ff-11ea-b3de-0242ac130004

Description:

Provides the characteristic to communicate selections to the output device.

Properties:

Write, Notify

3.14.20 Service: ScoreOutput

UUID: **d3560300-b9ff-11ea-b3de-0242ac130004

Description:

Provides an output-scoring service, which consumes fitted model parameters and generates a stream of stimulus scores.

Characteristic: **OutputScore**

UUID: **d3560301-b9ff-11ea-b3de-0242ac130004

Description:

Provides the characteristic to communicate the stimulus output score by sending: OUTPUTSCORE messages.

Properties:

Read, Notify.

Payload Format:

Read, Notify: encode the current output score as an OUTPUTSCORE message.

Characteristic: CurrentModel:

UUID: **d3560302-b9ff-11ea-b3de-0242ac130004

Description:

Provides a characteristic to communicate the current BCI model to the score-output module.

Properties:

Write

Payload Format:

Write: encode the current model as a CURRENTMODEL message. Note: this is a *full* message with type, version, length encoding (Later we may remove this technically unnecessary header information), thus it *will* take *more than* 1 message to communicate a full stimulus state when we have > 8 objectIDs on a single device.

Characteristic: CurrentSOSFILTER:

UUID: **d3560303-b9ff-11ea-b3de-0242ac130004

Description:

Provides a characteristic to communicate the current BCI filter to use.

Properties:

Write

PayloadFormat:

Write: Encode the IIR as a Second-Order-Sections filter, which is a matrix of 6 x n-Sections. Thus, we send it as such a matrix.

3.15 Print Your Own Headset

Find the files on Github, and a quick tutorial on YouTube of V1.0 made by Nikolai Schauer. Please read the printing guide before trying to print.

3.15.1 Parts

Per headset, 3 pieces need to be printed:

- 1x Main-Part.stl
- 2x Attachment.stl

We recommend semi-flexible filament, but anything flexible should work. Stiff material like PLA, PETG or ABS won't work! We used "Renkforce Flexible Filament" for all prototypes.

3.15.2 Print settings

We recommend these print settings:

- Nozzle: 0.4 mm
- Extrusionwidth: 0.5mm
- Layerheight: 0.25mm Brim: 3mm
- Speed: 20mm/s
- Infill: 40% (This value can be adjusted to compensate for more flexible filament; a higher infill value will make for a more stiff headset)
- Perimeters: 5
- Bottom layers: 5
- Top layers: 5
- Temperature: See filament spool

The print settings are of course dependent on your personal printer and will most likely require some tuning. The parts were designed to fit on most buildplates, in some cases "Main-Part.stl" needs to be rotated by a few degrees (Prusa MK3 for example).

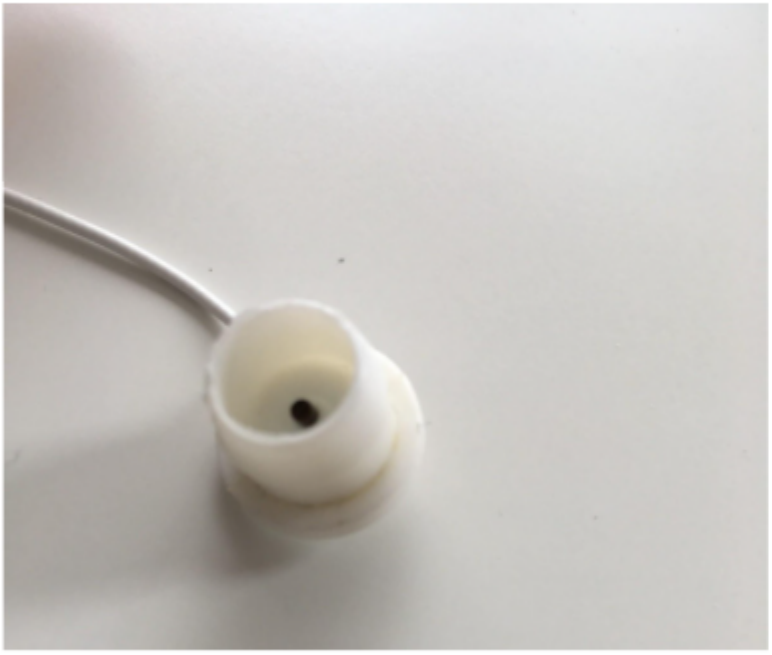
3.16 Set-up & fitting of MindAffect water electrodes

3.16.1 Set-up

To prepare the headset:

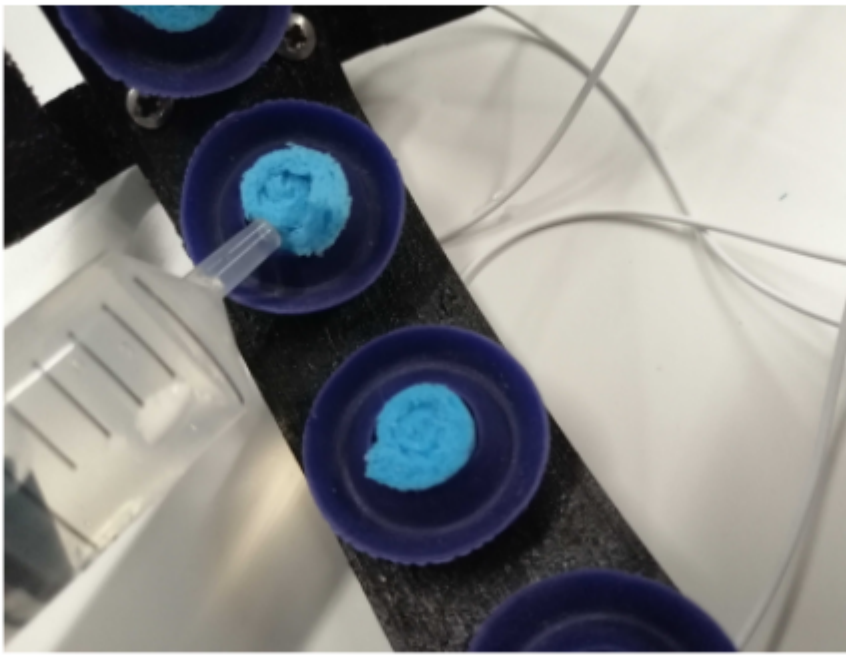
1. Get a glass of water
2. Fill the plastic syringe with water
3. Roll the sponges to fit into the headset
4. Put the sponges in the electrode cups in the headset.
5. Wet the sponges from the top with the syringe with water. Make sure to fill them until they are saturated. This can be detected when water doesn't get absorbed by the sponges anymore.

1)



2)

3)



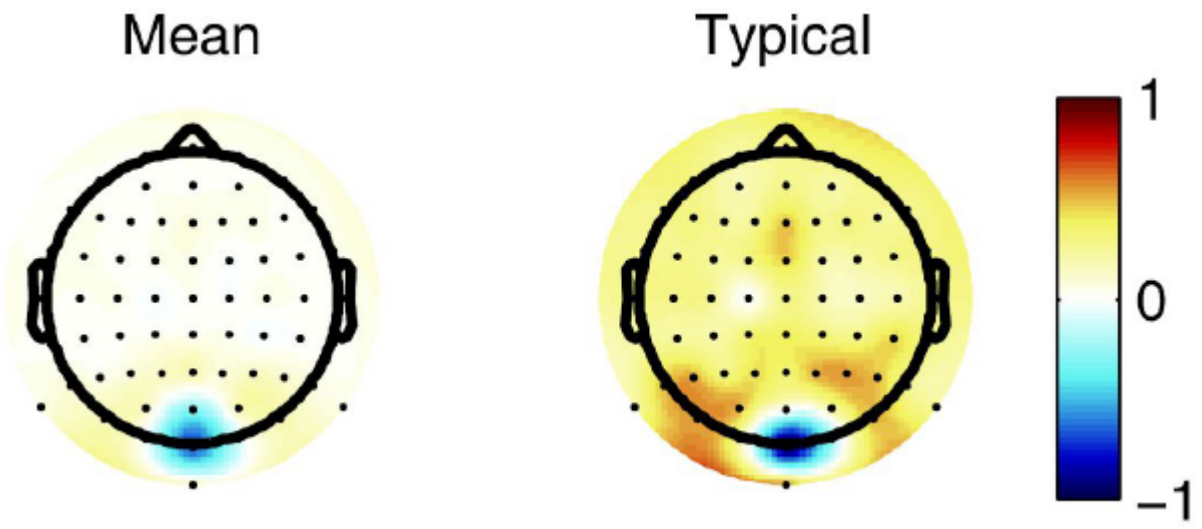
3.16.2 Fitting

Before positioning the headset on the user's head, inspect it to see if there are no damaged parts that might hurt the user. Please remove any metal parts in the hair (hair clips, ornaments, etc). Please note that the picture on the left contains an older version of the headset. The newer versions have the logo displayed on the left instead of the outer left electrode in the images. When using the system with several users or when the headset has become soiled, you can clean the headset using a moistened (by water) cloth. Do not use detergents, as this might damage the electrodes. The headset needs to be placed directly above the inion (a small protrusion at the back of the skull).

- Put the hard part of the headset on the back of the head with the electrode cables facing downwards (see image on the left upper corner).
- Tighten the strap on the front of the head to make sure the headset stays in place (not too tight that it gets uncomfortable).
- Press all electrodes carefully on the head by pressing the white circle plates. The participant must feel it is wet.
- If the cables were disconnected from the green connector plate, please connect the cables to the right number displayed next to the plugs on the green connector
- Connect the connector to the OpenBCI amplifier with 'BCI top' test facing upwards (such that channel 2,4 and ground are facing upwards as well).

3.17 Headset layout

Brain-Computer Interfaces (BCIs) allow users to control devices and communicate by using brain activity only. BCIs based on broad-band visual stimulation can outperform BCIs using other stimulation paradigms. Visual stimulation with pseudo-random bit-sequences evokes specific code-modulated Visual Evoked Potentials (cVEPs) that can be reliably used in BCI for high-speed communication in, for example, speller applications. The typical response to code-modulated Visual Evoked Potential Brain Computer Interfacing is strongest in the locations as shown below, where the blue spot has the significant activation.



https://doi.org/10.1371/journal.pone.0133797

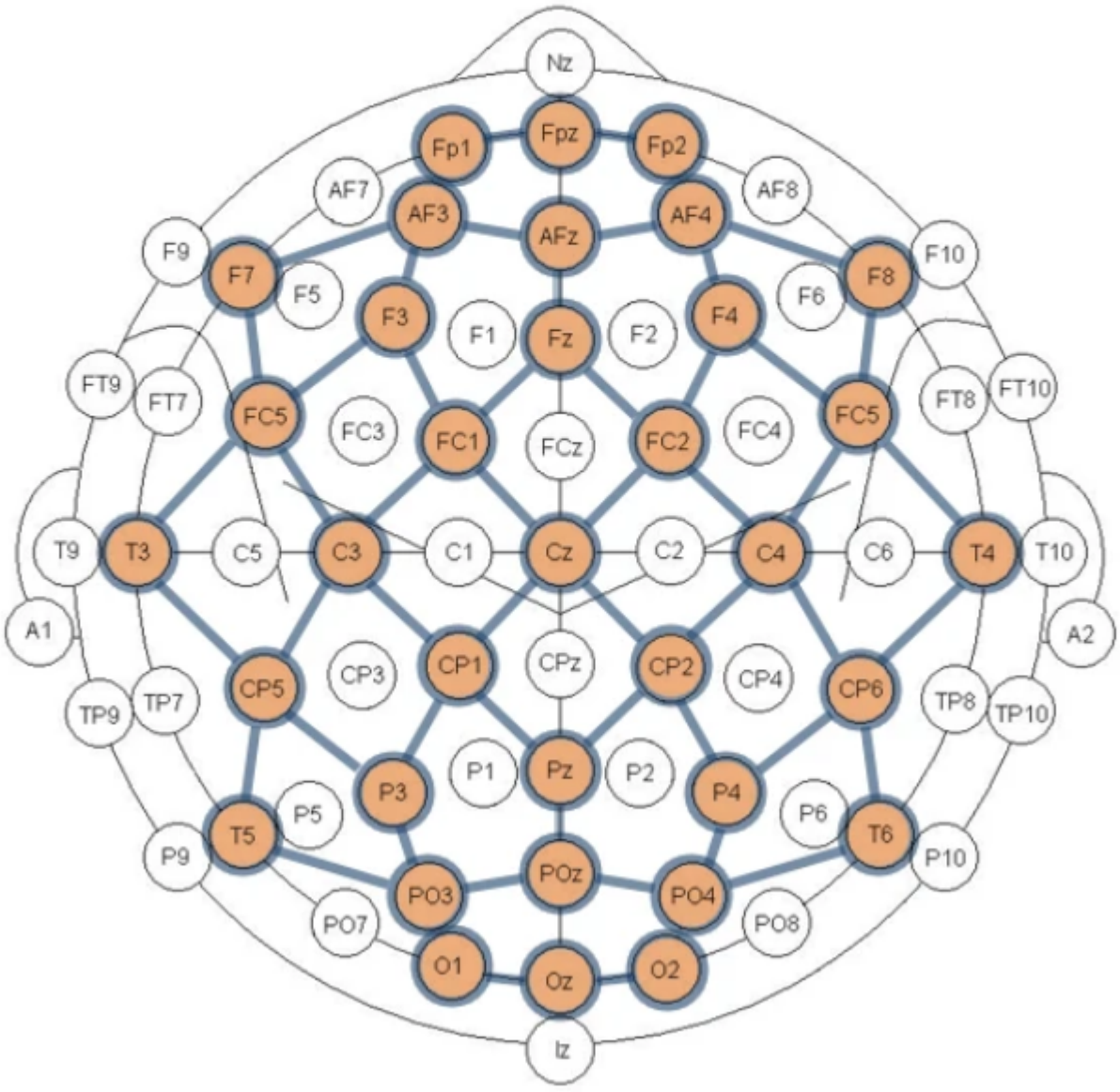
Our headset is designed such that the bottom electrode is just above the inion, the bump on the back of your head right above the neck. The electrodes are spaced exactly 4cm apart, this distance is based on extensive data analysis of acquired EEG signals. The headset used by MindAffect contains water-based electrodes with AgCl. However, we believe it should work with typical cup-electrodes or dry electrodes when the connections are made right.

3.17.1 MindAffect headset

For the Ganglion we only use 4 working electrodes, located in a row approximately 1cm above the inion. They are centered around the reference electrode in the middle (median/sagittal plane), which leads to two electrodes on the left and two on the right of the reference electrode. The ground is mounted on the forehead by a conducting headband, however, one can use an electrode sticker on the forehead, a bracelet around the wrist or use an electrode as far away as possible from the brain signals measured.

3.17.2 MarkIV UltraCortex headset

The OpenBCI headset has successfully been tested with the use of the O1, Oz, O2, PO3, POz, PO4, T5, ad T6 electrodes for the signals, one of the earclips for reference and the other earclip for the Ground. Consult the image below for the respective locations on the headset. Please note that only a handful of tests have been conducted with this headset, and the electrodes might not be optimal. Also, note that the headset uses dry electrodes, this might induce longer classification times.


Ultracortex Mark IV
Node Locations (35 total)
Nz
Fp1
Fpz
Fp2
AF7
AF3
AFz
AF4
AF8
F9
F7
F5
F3
F1
Fz
F2
F4
F6
F8
F10
FT9
FT7
FC5
FC3
FC1
FCz
FC2
FC4
FC5
FT8
FT10
T9
T3
C5
C3
C1
Cz
C2
C4
C6
T4
T10
A1
A2
TP9
TP7
CP5
CP3
CP1
CPz
CP2
CP4
CP6
TP8
TP10
P9
T5
P5
P3
P1
Pz
P2
P4
P6
T6
P10
PO7
PO3
POz
PO4
PO8
O1
Oz
O2
Iz

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

mindaffectBCI, 159
mindaffectBCI.decoder, 131
mindaffectBCI.decoder.analyse_datasets, 88
mindaffectBCI.decoder.ccaViewer, 91
mindaffectBCI.decoder.datasets, 92
mindaffectBCI.decoder.decoder, 93
mindaffectBCI.decoder.decodingCurveSupervised, 97
mindaffectBCI.decoder.decodingSupervised, 99
mindaffectBCI.decoder.devent2stimsequence, 100
mindaffectBCI.decoder.erpViewer, 101
mindaffectBCI.decoder.FileProxyHub, 79
mindaffectBCI.decoder.lower_bound_tracker, 101
mindaffectBCI.decoder.model_fitting, 102
mindaffectBCI.decoder.multipleCCA, 106
mindaffectBCI.decoder.multipleMCCA, 107
mindaffectBCI.decoder.normalizeOutputScores, 108
mindaffectBCI.decoder.normalizeOutputScores_streamed, 111
mindaffectBCI.decoder.offline, 78
mindaffectBCI.decoder.offline.load_brainsonfire, 69
mindaffectBCI.decoder.offline.load_brainstream, 70
mindaffectBCI.decoder.offline.load_cocktail, 70
mindaffectBCI.decoder.offline.load_mark_EMG, 71
mindaffectBCI.decoder.offline.load_mindaffectBCI, 72
mindaffectBCI.decoder.offline.load_mTRF_audio, 71
mindaffectBCI.decoder.offline.load_ninapro_db2, 73
mindaffectBCI.decoder.offline.load_openBMI, 73
mindaffectBCI.decoder.offline.load_p300_prn, 75
mindaffectBCI.decoder.offline.load_twofinger, 75
mindaffectBCI.decoder.offline.read_buffer_offline, 75
mindaffectBCI.decoder.offline.read_mindaffectBCI, 76
mindaffectBCI.decoder.preprocess, 111
mindaffectBCI.decoder.readCapInf, 114
mindaffectBCI.decoder.scoreOutput, 115
mindaffectBCI.decoder.scoreStimulus, 116
mindaffectBCI.decoder.sigViewer, 117
mindaffectBCI.decoder.startUtopiaHub, 117
mindaffectBCI.decoder.stim2event, 118
mindaffectBCI.decoder.timestamp_check, 118
mindaffectBCI.decoder.trigger_check, 118
mindaffectBCI.decoder.updateSummaryStatistics, 119
mindaffectBCI.decoder.utils, 126
mindaffectBCI.decoder.UtopiaDataInterface, 79
mindaffectBCI.decoder.zscore2Ptgt_softmax, 129
mindaffectBCI.noisetag, 131
mindaffectBCI.online_bci, 139
mindaffectBCI.ssdpDiscover, 141
mindaffectBCI.stimseq, 144
mindaffectBCI.utopia2output, 146
mindaffectBCI.utopiaclient, 149
mindaffectBCI.utopiaController, 147

Index

A

add_sample_timestamps() (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface method*), 80
addMessageHandler() (*mindaffectBCI.noisetag.Noisetag method*), 134
addMessageHandler() (*mindaffectBCI.utopiaController.UtopiaController method*), 147
addpoint() (*mindaffectBCI.noisetag.sumstats method*), 139
addPredictionHandler() (*mindaffectBCI.noisetag.Noisetag method*), 134
addPredictionHandler() (*mindaffectBCI.utopiaController.UtopiaController method*), 147
addSelectionHandler() (*mindaffectBCI.noisetag.Noisetag method*), 134
addSelectionHandler() (*mindaffectBCI.utopiaController.UtopiaController method*), 147
addSignalQualityHandler() (*mindaffectBCI.utopiaController.UtopiaController method*), 147
addSubscription() (*mindaffectBCI.noisetag.Noisetag method*), 134
addSubscription() (*mindaffectBCI.utopiaController.UtopiaController method*), 147
analyse_dataset() (*in module mindaffectBCI.decoder.analyse_datasets*), 88
analyse_datasets() (*in module mindaffectBCI.decoder.analyse_datasets*), 88
analyse_train_test() (*in module mindaffectBCI.decoder.analyse_datasets*), 89
append() (*mindaffectBCI.decoder.lower_bound_tracker.lower_bound_tracker method*), 101
append() (*mindaffectBCI.decoder.utils.RingBuffer method*), 126
argsort() (*in module mindaffectBCI.decoder.offline.load_cocktail*), 70
audc_score() (*mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence static method*), 102
autoconnect() (*mindaffectBCI.decoder.FileProxyHub.FileProxyHub method*), 79
autoconnect() (*mindaffectBCI.utopiaclient.UtopiaClient method*), 156
autoconnect() (*mindaffectBCI.utopiaController.UtopiaController method*), 147
autocov() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 119

B

BaseEstimator (*class in mindaffectBCI.decoder.model_fitting*), 102
BaseSequence2Sequence (*class in mindaffectBCI.decoder.model_fitting*), 102
block_randomize() (*in module mindaffectBCI.decoder.utils*), 126
brains_on_fire_online() (*in module mindaffectBCI.decoder.datasets*), 92
butter_filterbank() (*in module mindaffectBCI.decoder.preprocess*), 111
butter_sosfilt() (*in module mindaffectBCI.decoder.utils*), 127
butterfilt_and_downsample (*class in mindaffectBCI.decoder.UtopiaDataInterface*), 83
BwdLinearRegression (*class in mindaffectBCI.decoder.model_fitting*), 104

C

c4() (*in module mindaffectBCI.decoder.normalizeOutputScores*), 108

calibrate_softmaxscale() (*in module mindaffectBCI.decoder.zscore2Ptgt_softmax*), 129
CalibrationPhase (*class in mindaffectBCI.noisetag*), 131
ccaViewer() (*in module mindaffectBCI.decoder.ccaViewer*), 91
channel_power_standardizer (*class in mindaffectBCI.decoder.UtopiaDataInterface*), 84
check_is_running() (*in module mindaffectBCI.online_bci*), 139
ClassifierMixin (*class in mindaffectBCI.decoder.model_fitting*), 104
clear() (*mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence method*), 102
clear() (*mindaffectBCI.decoder.utils.RingBuffer method*), 126
clear() (*mindaffectBCI.noisetag.GSM method*), 133
clearLastPrediction() (*mindaffectBCI.noisetag.Noisetag method*), 134
clearLastPrediction() (*mindaffectBCI.utopiaController.UtopiaController method*), 147
clearLastSelection() (*mindaffectBCI.noisetag.Noisetag method*), 134
clearLastSignalQuality() (*mindaffectBCI.noisetag.Noisetag method*), 134
cocktail() (*in module mindaffectBCI.decoder.datasets*), 92
combine_Ptgt() (*in module mindaffectBCI.decoder.decoder*), 93
compCxx_diag() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 120
compCxx_full() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 120
compCyx_diag() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 120
compCyx_full() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 120
compCyy_diag() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 121
compCyy_diag_perY() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 121
compCyy_full() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 121
compute_decoding_curve() (*in module mindaffectBCI.decoder.decodingCurveSupervised*), 97
compute_pval_curve() (*in module mindaffectBCI.decoder.normalizeOutputScores_streamed*), 111
compute_softmax_curve() (*in module mindaffectBCI.decoder.normalizeOutputScores_streamed*), 111
compute_stopping_curve() (*in module mindaffectBCI.decoder.decodingCurveSupervised*), 97
connect() (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface method*), 80
connect() (*mindaffectBCI.noisetag.Noisetag method*), 134
connect() (*mindaffectBCI.utopia2output.Utopia2Output method*), 146
connect() (*mindaffectBCI.utopiaclient.UtopiaClient method*), 156
convertstimSeq2int() (*mindaffectBCI.stimseq.StimSeq method*), 144
convWX() (*in module mindaffectBCI.decoder.scoreOutput*), 115
convXYR() (*in module mindaffectBCI.decoder.scoreOutput*), 115
convYR() (*in module mindaffectBCI.decoder.scoreOutput*), 115
cov() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 121
crossautocov() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 122
cv_fit() (*mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence method*), 102
cv_fit() (*mindaffectBCI.decoder.model_fitting.MultiCCA method*), 105
cvSupervised() (*in module mindaffectBCI.decoder.multipleCCA*), 106
cvSupervised() (*in module mindaffectBCI.decoder.multipleMCCA*), 107
Cxx_diag2full() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 119
Cyx_diag2full() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 119
Cyy_diag2full() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 119
Cyy_tyeye_diag2full() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 119
Cyy_yetet_diag2full() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 119

D

DataHeader (*class in mindaffectBCI.utopiaclient*), 149
DataPacket (*class in mindaffectBCI.utopiaclient*), 150
datapackets2array() (*in module mindaffectBCI.decoder.offline.read_mindaffectBCI*), 76
dataset_generator() (*in module mindaffectBCI.decoder.datasets*), 92
dataset_to_XY_ndarrays() (*in module mindaffectBCI.decoder.decoder*), 93

datasettest() (*in module mindaffectBCI.decoder.scoreOutput*), 115
debug_test_dataset() (*in module mindaffectBCI.decoder.analyse_datasets*), 89
debug_test_single_dataset() (*in module mindaffectBCI.decoder.analyse_datasets*), 90
decode_proba() (*mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence method*), 103
decodeRawMessage() (*in module mindaffectBCI.utopiaclient*), 158
decodeRawMessages() (*in module mindaffectBCI.utopiaclient*), 159
decodingCurveSupervised() (*in module mindaffectBCI.decoder.decodingCurveSupervised*), 97
decodingSupervised() (*in module mindaffectBCI.decoder.decodingSupervised*), 99
decodingSupervised_streamed() (*in module mindaffectBCI.decoder.decodingSupervised*), 100
dedupY0() (*in module mindaffectBCI.decoder.scoreOutput*), 115
DEFAULTHOST (*mindaffectBCI.utopiaclient.UtopiaClient attribute*), 156
DEFAULTPORT (*mindaffectBCI.utopiaclient.UtopiaClient attribute*), 156
deserialize() (*mindaffectBCI.utopiaclient.DataHeader static method*), 149
deserialize() (*mindaffectBCI.utopiaclient.DataPacket static method*), 150
deserialize() (*mindaffectBCI.utopiaclient.Heartbeat static method*), 150
deserialize() (*mindaffectBCI.utopiaclient.Log static method*), 151
deserialize() (*mindaffectBCI.utopiaclient.ModeChange static method*), 151
deserialize() (*mindaffectBCI.utopiaclient.NewTarget static method*), 152
deserialize() (*mindaffectBCI.utopiaclient.PredictedTargetDist static method*), 152
deserialize() (*mindaffectBCI.utopiaclient.PredictedTargetProb static method*), 153
deserialize() (*mindaffectBCI.utopiaclient.RawMessage class method*), 153
deserialize() (*mindaffectBCI.utopiaclient.Reset static method*), 154
deserialize() (*mindaffectBCI.utopiaclient.Selection static method*), 154
deserialize() (*mindaffectBCI.utopiaclient.SignalQuality static method*), 154
deserialize() (*mindaffectBCI.utopiaclient.StimulusEvent static method*), 155
deserialize() (*mindaffectBCI.utopiaclient.Subscribe static method*), 155
deserializeMany() (*mindaffectBCI.utopiaclient.RawMessage class method*), 153
devent2stimSequence() (*in module mindaffectBCI.decoder.devent2stimsequence*), 100
disableHeartbeats() (*mindaffectBCI.utopiaclient.UtopiaClient method*), 157
disconnect() (*mindaffectBCI.utopiaclient.UtopiaClient method*), 157
discover() (*mindaffectBCI.ssdpDiscover.ssdpDiscover method*), 142
discoverOrIPscan() (*in module mindaffectBCI.ssdpDiscover*), 141
doCalibrationSupervised() (*in module mindaffectBCI.decoder.decoder*), 93
doFrame() (*in module mindaffectBCI.noisetag*), 138
doModelFitting() (*in module mindaffectBCI.decoder.decoder*), 93
doOutput() (*mindaffectBCI.utopia2output.Utopia2Output method*), 146
doPrediction() (*in module mindaffectBCI.decoder.decoder*), 94
doPredictionStatic() (*in module mindaffectBCI.decoder.decoder*), 94

E

enableHeartbeats() (*mindaffectBCI.utopiaclient.UtopiaClient method*), 157
entropy() (*in module mindaffectBCI.decoder.zscore2Ptgt_softmax*), 130
equals_subarray() (*in module mindaffectBCI.decoder.utils*), 127
erpViewer() (*in module mindaffectBCI.decoder.erpViewer*), 101
estimate_Fy_noise_variance()

(*in module mindaffectBCI.decoder.normalizeOutputScores*), 108
estimate_Fy_noise_variance_2() (*in module mindaffectBCI.decoder.normalizeOutputScores*), 109
eventSeq (*mindaffectBCI.stimseq.StimSeq attribute*), 144
Experiment (*class in mindaffectBCI.noisetag*), 132
extend() (*mindaffectBCI.decoder.utils.RingBuffer method*), 126
extract_data_segment() (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface method*), 80
extract_envelope() (*in module mindaffectBCI.decoder.preprocess*), 111
extract_msgs_segment() (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface method*), 81
extract_ringbuffer_segment() (*in module mindaffectBCI.decoder.utils*), 127
extract_stimulus_segment() (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface method*), 81

F

factored2full() (*in module mindaffectBCI.decoder.scoreStimulus*), 116
fft_filterbank() (*in module mindaffectBCI.decoder.preprocess*), 112
FileProxyHub (*class in mindaffectBCI.decoder.FileProxyHub*), 79
filter_Fy() (*in module mindaffectBCI.decoder.normalizeOutputScores*), 109
filterbank_testcase() (*in module mindaffectBCI.decoder.multipleCCA*), 106
fir() (*in module mindaffectBCI.decoder.preprocess*), 112
fit() (*mindaffectBCI.decoder.lower_bound_tracker.lower_bound_tracker method*), 101
fit() (*mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence method*), 103
fit() (*mindaffectBCI.decoder.model_fitting.BwdLinearRegression method*), 104
fit() (*mindaffectBCI.decoder.model_fitting.FwdLinearRegression method*), 105
fit() (*mindaffectBCI.decoder.model_fitting.LinearSklearn method*), 105
fit() (*mindaffectBCI.decoder.model_fitting.MultiCCA method*), 105
fit() (*mindaffectBCI.decoder.UtopiaDataInterface.butterfilt_and_downsample method*), 83
fit() (*mindaffectBCI.decoder.UtopiaDataInterface.channel_power method*), 84
fit() (*mindaffectBCI.decoder.UtopiaDataInterface.power_tracker method*), 84
fit() (*mindaffectBCI.decoder.UtopiaDataInterface.stim2eventfilt method*), 85
fit() (*mindaffectBCI.decoder.UtopiaDataInterface.temporal_decorrelato method*), 86
fit() (*mindaffectBCI.decoder.UtopiaDataInterface.timestamp_interpolat method*), 87
fit() (*mindaffectBCI.decoder.UtopiaDataInterface.TransformerMixin method*), 79
fit_b() (*mindaffectBCI.decoder.model_fitting.MultiCCA method*), 105
fit_predict() (*in module mindaffectBCI.decoder.multipleCCA*), 106
flatten_decoding_curves() (*in module mindaffectBCI.decoder.analyse_datasets*), 91
Flicker (*class in mindaffectBCI.noisetag*), 132
FlickerWithSelection (*class in mindaffectBCI.noisetag*), 132
float2int() (*mindaffectBCI.stimseq.StimSeq static method*), 144
fromFile() (*mindaffectBCI.stimseq.StimSeq static method*), 144
fromString() (*mindaffectBCI.stimseq.StimSeq static method*), 144
fromUtopiaMessage() (*mindaffectBCI.utopiaclient.RawMessage class method*), 153
FSM (*class in mindaffectBCI.noisetag*), 132
ftevent (*class in mindaffectBCI.decoder.offline.read_buffer_offline*), 75
ftheader (*class in mindaffectBCI.decoder.offline.read_buffer_offline*), 75
FwdLinearRegression (*class in mindaffectBCI.decoder.model_fitting*), 104

G

get() (*mindaffectBCI.noisetag.Flicker method*), 132
get() (*mindaffectBCI.noisetag.FSM method*), 132
get() (*mindaffectBCI.noisetag.GSM method*), 133
get() (*mindaffectBCI.noisetag.WaitFor method*), 138
get_all_ips() (*in module mindaffectBCI.ssdpDiscover*), 141
get_dataset() (*in module mindaffectBCI.decoder.datasets*), 92
get_ip_address() (*in module mindaffectBCI.ssdpDiscover*), 142
get_local_ip() (*in module mindaffectBCI.ssdpDiscover*), 142
get_remote_ip() (*in module mindaffectBCI.ssdpDiscover*), 142

get_trial_start_end() (*in module mindaffectBCI.decoder.decoder*), 95
get_trl_ep_idx() (*in module mindaffectBCI.decoder.offline.load_openBMI*), 73
get_valid_epochs_outputs() (*in module mindaffectBCI.decoder.normalizeOutputScores*), 109
getCalibration_dataset() (*in module mindaffectBCI.decoder.decoder*), 94
gethostport() (*mindaffectBCI.noisetag.Noisetag method*), 135
gethostport() (*mindaffectBCI.utopiaclient.UtopiaClient method*), 157
gethostport() (*mindaffectBCI.utopiaController.UtopiaController method*), 148
getLastPrediction() (*mindaffectBCI.noisetag.Noisetag method*), 134
getLastPrediction() (*mindaffectBCI.utopiaController.UtopiaController method*), 147
getLastSelection() (*mindaffectBCI.noisetag.Noisetag method*), 135
getLastSelection() (*mindaffectBCI.utopiaController.UtopiaController method*), 147
getLastSignalQuality() (*mindaffectBCI.noisetag.Noisetag method*), 135
getLastSignalQuality() (*mindaffectBCI.utopiaController.UtopiaController method*), 148
getNewMessages() (*mindaffectBCI.decoder.FileProxyHub.FileProxyHub method*), 79
getNewMessages() (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface method*), 81
getNewMessages() (*mindaffectBCI.noisetag.Noisetag method*), 135
getNewMessages() (*mindaffectBCI.utopiaclient.UtopiaClient method*), 157
getNewMessages() (*mindaffectBCI.utopiaController.UtopiaController method*), 148
getPosInfo() (*in module mindaffectBCI.decoder.readCapInf*), 114
getStimulusState() (*mindaffectBCI.noisetag.Noisetag method*), 135
getTimeStamp() (*mindaffectBCI.decoder.FileProxyHub.FileProxyHub method*), 79
getTimeStamp() (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface method*), 81
getTimeStamp() (*mindaffectBCI.noisetag.Noisetag method*), 135
getTimeStamp() (*mindaffectBCI.utopiaclient.TimeStampClock method*), 156
getTimeStamp() (*mindaffectBCI.utopiaclient.UtopiaClient method*), 157
getTimeStamp() (*mindaffectBCI.utopiaController.UtopiaController method*), 148
getX() (*mindaffectBCI.decoder.lower_bound_tracker.lower_bound_tracke method*), 101
getY() (*mindaffectBCI.decoder.lower_bound_tracker.lower_bound_tracke method*), 101
GSM (*class in mindaffectBCI.noisetag*), 133

H

Heartbeat (*class in mindaffectBCI.utopiaclient*), 150
HEARTBEATINTERVAL_ms (*mindaffectBCI.utopiaclient.UtopiaClient attribute*), 156
HEARTBEATINTERVALUDP_ms (*mindaffectBCI.utopiaclient.UtopiaClient attribute*), 156
HighlightObject (*class in mindaffectBCI.noisetag*), 133
hist() (*mindaffectBCI.noisetag.sumstats method*), 139
hl2alpha() (*mindaffectBCI.decoder.UtopiaDataInterface.power_tracker method*), 85

I

idOutliers() (*in module mindaffectBCI.decoder.utils*), 127
iir_sosfilt_sos() (*in module mindaffectBCI.decoder.utils*), 127
incremental_estimate_noise_variance() (*in module mindaffectBCI.decoder.normalizeOutputScores_streamed*), 111
initClockAlign() (*mindaffectBCI.utopiaclient.UtopiaClient method*), 157
initDataRingBuffer() (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface method*), 81
initSocket() (*mindaffectBCI.ssdpDiscover.ssdpDiscover method*), 143
initStimulusRingBuffer() (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface*

method), 81
injectERP() (*in module mindaffectBCI.utopiaController*), 149
ip_is_local() (*in module mindaffectBCI.ssdpDiscover*), 142
ipscanDiscover() (*in module mindaffectBCI.ssdpDiscover*), 142
is_alive() (*mindaffectBCI.online_bci.NoneProc method*), 139
is_fitted() (*mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence method*), 103
isConnected() (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface method*), 81
isConnected() (*mindaffectBCI.noisetag.Noisetag method*), 135
isConnected() (*mindaffectBCI.utopiaController.UtopiaController method*), 148

J

join() (*mindaffectBCI.online_bci.NoneProc method*), 139

L

lab2ind() (*in module mindaffectBCI.decoder.utils*), 127
latlong2xy() (*in module mindaffectBCI.decoder.readCapInf*), 115
latlong2xyz() (*in module mindaffectBCI.decoder.readCapInf*), 115
LinearSklearn (*class in mindaffectBCI.decoder.model_fitting*), 105
load_brainsonfire() (*in module mindaffectBCI.decoder.offline.load_brainsonfire*), 69
load_brainstream() (*in module mindaffectBCI.decoder.offline.load_brainstream*), 70
load_cocktail() (*in module mindaffectBCI.decoder.offline.load_cocktail*), 70
load_config() (*in module mindaffectBCI.online_bci*), 139
load_mark_EMG() (*in module mindaffectBCI.decoder.offline.load_mark_EMG*), 71
load_mindaffectBCI() (*in module mindaffectBCI.decoder.offline.load_mindaffectBCI*), 72
load_mTRF_audio() (*in module mindaffectBCI.decoder.offline.load_mTRF_audio*), 71
load_ninapro_db2() (*in module mindaffectBCI.decoder.offline.load_ninapro_db2*), 73
load_openBMI() (*in module mindaffectBCI.decoder.offline.load_openBMI*), 73
load_p300_prn() (*in module mindaffectBCI.decoder.offline.load_p300_prn*), 75
load_previous_dataset() (*in module mindaffectBCI.decoder.decoder*), 95
load_twofinger() (*in module mindaffectBCI.decoder.offline.load_twofinger*), 75
local_slope() (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface method*), 81
Log (*class in mindaffectBCI.utopiaclient*), 151
log() (*mindaffectBCI.noisetag.Noisetag method*), 135
log() (*mindaffectBCI.utopiaController.UtopiaController method*), 148
lower_bound_tracker (*class in mindaffectBCI.decoder.lower_bound_tracker*), 101
lowerbands() (*in module mindaffectBCI.decoder.datasets*), 92

M

make_ssvep_ref_signals() (*in module mindaffectBCI.decoder.offline.load_openBMI*), 74
makeDiscoveryMessage() (*mindaffectBCI.ssdpDiscover.ssdpDiscover method*), 143
marginalize_scores() (*in module mindaffectBCI.decoder.zscore2Ptgt_softmax*), 130
mark_EMG() (*in module mindaffectBCI.decoder.datasets*), 92
MAXMESSAGESIZE (*mindaffectBCI.utopiaclient.UtopiaClient attribute*), 156
mean() (*mindaffectBCI.decoder.UtopiaDataInterface.power_tracker method*), 85
messagelogger() (*mindaffectBCI.utopiaclient.UtopiaClient method*), 157
messagePingPong() (*mindaffectBCI.utopiaclient.UtopiaClient method*), 157
mindaffectBCI (*module*), 159
mindaffectBCI() (*in module mindaffectBCI.decoder.datasets*), 92
mindaffectBCI.decoder (*module*), 131
mindaffectBCI.decoder.analyse_datasets (*module*), 88
mindaffectBCI.decoder.ccaViewer (*module*), 91
mindaffectBCI.decoder.datasets (*module*), 92
mindaffectBCI.decoder.decoder (*module*), 93
mindaffectBCI.decoder.decodingCurveSupervised (*module*), 97
mindaffectBCI.decoder.decodingSupervised (*module*), 99
mindaffectBCI.decoder.devent2stimsequence (*module*), 100

mindaffectBCI.decoder.erpViewer (*module*), 101
mindaffectBCI.decoder.FileProxyHub (*module*), 79
mindaffectBCI.decoder.lower_bound_tracker (*module*), 101
mindaffectBCI.decoder.model_fitting (*module*), 102
mindaffectBCI.decoder.multipleCCA (*module*), 106
mindaffectBCI.decoder.multipleMCCA (*module*), 107
mindaffectBCI.decoder.normalizeOutputScores (*module*), 108
mindaffectBCI.decoder.normalizeOutputScores_affineCal (*module*), 111
mindaffectBCI.decoder.offline (*module*), 78
mindaffectBCI.decoder.offline.load_brainflow (*module*), 69
mindaffectBCI.decoder.offline.load_brainstream (*module*), 70
mindaffectBCI.decoder.offline.load_cocktail (*module*), 70
mindaffectBCI.decoder.offline.load_mark_mindaffectBCI (*module*), 71
mindaffectBCI.decoder.offline.load_mindaffectSSD (*module*), 72
mindaffectBCI.decoder.offline.load_mTRF_audio (*module*), 71
mindaffectBCI.decoder.offline.load_ninapro_db2 (*module*), 73
mindaffectBCI.decoder.offline.load_openBMI (*module*), 73
mindaffectBCI.decoder.offline.load_p300_prob (*module*), 75
mindaffectBCI.decoder.offline.load_twofinger (*module*), 75
mindaffectBCI.decoder.offline.read_buffer_offline (*module*), 75
mindaffectBCI.decoder.offline.read_mindaffectBCI (*module*), 76
mindaffectBCI.decoder.preprocess (*module*), 111
mindaffectBCI.decoder.readCapInf (*module*), 114
mindaffectBCI.decoder.scoreOutput (*module*), 115
mindaffectBCI.decoder.scoreStimulus (*module*), 116
mindaffectBCI.decoder.sigViewer (*module*), 117
mindaffectBCI.decoder.startUtopiaHub (*module*), 117
mindaffectBCI.decoder.stim2event (*module*), 118
mindaffectBCI.decoder.timestamp_check (*module*), 118
mindaffectBCI.decoder.trigger_check (*module*), 118
mindaffectBCI.decoder.updateSummaryStatistics (*module*), 119
mindaffectBCI.decoder.utils (*module*), 126
mindaffectBCI.decoder.UtopiaDataInterface (*module*), 79
mindaffectBCI.decoder.zscore2Ptgt_softmax (*module*), 129
mindaffectBCI.noisetag (*module*), 131
mindaffectBCI.online_bci (*module*), 139
mindaffectBCI.ssdpDiscover (*module*), 141
mindaffectBCI.stimseq (*module*), 144
mindaffectBCI.utopia2output (*module*), 146
mindaffectBCI.utopiaclient (*module*), 149
mindaffectBCI.utopiaController (*module*), 147
mkBlinkingSequence() (*in module mindaffectBCI.noisetag*), 138
mkCodes() (*in module mindaffectBCI.stimseq*), 145
mkFreqTag() (*in module mindaffectBCI.stimseq*), 145
mkRowCol() (*in module mindaffectBCI.stimseq*), 145
mkStimEvent() (*mindaffectBCI.utopiaController.UtopiaController method*), 148
mktestFy() (*in module mindaffectBCI.decoder.normalizeOutputScores*), 109
mkTestFy() (*in module mindaffectBCI.decoder.zscore2Ptgt_softmax*), 130
ModeChange (*class in mindaffectBCI.utopiaclient*), 151
modeChange() (*mindaffectBCI.noisetag.Noisetag method*), 135
modeChange() (*mindaffectBCI.utopiaController.UtopiaController method*), 148
msearchTemplate (*mindaffectBCI.ssdpDiscover.ssdpDiscover attribute*), 143
msgID (*mindaffectBCI.utopiaclient.DataHeader attribute*), 150
msgID (*mindaffectBCI.utopiaclient.DataPacket attribute*), 150
msgID (*mindaffectBCI.utopiaclient.Heartbeat attribute*), 151
msgID (*mindaffectBCI.utopiaclient.Log attribute*), 151
msgID (*mindaffectBCI.utopiaclient.ModeChange attribute*), 151
msgID (*mindaffectBCI.utopiaclient.NewTarget attribute*), 152
msgID (*mindaffectBCI.utopiaclient.PredictedTargetDist attribute*), 152

msgID (*mindaffectBCI.utopiaclient.PredictedTargetProb attribute*), 153
msgID (*mindaffectBCI.utopiaclient.Reset attribute*), 154
msgID (*mindaffectBCI.utopiaclient.Selection attribute*), 154
msgID (*mindaffectBCI.utopiaclient.SignalQuality attribute*), 155
msgID (*mindaffectBCI.utopiaclient.StimulusEvent attribute*), 155
msgID (*mindaffectBCI.utopiaclient.Subscribe attribute*), 155
msgName (*mindaffectBCI.utopiaclient.DataHeader attribute*), 150
msgName (*mindaffectBCI.utopiaclient.DataPacket attribute*), 150
msgName (*mindaffectBCI.utopiaclient.Heartbeat attribute*), 151
msgName (*mindaffectBCI.utopiaclient.Log attribute*), 151
msgName (*mindaffectBCI.utopiaclient.ModeChange attribute*), 151
msgName (*mindaffectBCI.utopiaclient.NewTarget attribute*), 152
msgName (*mindaffectBCI.utopiaclient.PredictedTargetDist attribute*), 152
msgName (*mindaffectBCI.utopiaclient.PredictedTargetProb attribute*), 153
msgName (*mindaffectBCI.utopiaclient.RawMessage attribute*), 153
msgName (*mindaffectBCI.utopiaclient.Reset attribute*), 154
msgName (*mindaffectBCI.utopiaclient.Selection attribute*), 154
msgName (*mindaffectBCI.utopiaclient.SignalQuality attribute*), 155
msgName (*mindaffectBCI.utopiaclient.StimulusEvent attribute*), 155
msgName (*mindaffectBCI.utopiaclient.Subscribe attribute*), 156
mTRF_audio() (*in module mindaffectBCI.decoder.datasets*), 92
MultiCCA (*class in mindaffectBCI.decoder.model_fitting*), 105
multipleCCA() (*in module mindaffectBCI.decoder.multipleCCA*), 106
multipleCCA() (*in module mindaffectBCI.decoder.multipleMCCA*), 107

N

newMessageHandler() (*in module mindaffectBCI.utopiaController*), 149
NewTarget (*class in mindaffectBCI.utopiaclient*), 152
newTarget() (*mindaffectBCI.utopiaController.UtopiaController method*), 148
next() (*mindaffectBCI.noisetag.CalibrationPhase method*), 131
next() (*mindaffectBCI.noisetag.Experiment method*), 132
next() (*mindaffectBCI.noisetag.Flicker method*), 132
next() (*mindaffectBCI.noisetag.FlickerWithSelection method*), 133
next() (*mindaffectBCI.noisetag.FSM method*), 132
next() (*mindaffectBCI.noisetag.GSM method*), 133
next() (*mindaffectBCI.noisetag.PredictionPhase method*), 137
next() (*mindaffectBCI.noisetag.SingleTrial method*), 138
next() (*mindaffectBCI.noisetag.WaitFor method*), 138
nic_info() (*in module mindaffectBCI.ssdpDiscover*), 142
ninapro_db2() (*in module mindaffectBCI.decoder.datasets*), 92
Noisetag (*class in mindaffectBCI.noisetag*), 133
NoneProc (*class in mindaffectBCI.online_bci*), 139
normalizeOutputScores() (*in module mindaffectBCI.decoder.normalizeOutputScores*), 109
normalizeOutputScores_streamed() (*in module mindaffectBCI.decoder.normalizeOutputScores_streamed*), 111
NotFittedError, 105

O

openBMI() (*in module mindaffectBCI.decoder.datasets*), 92

P

p300_prn() (*in module mindaffectBCI.decoder.datasets*), 92
parse_args() (*in module mindaffectBCI.decoder.ccaViewer*), 91
parse_args() (*in module mindaffectBCI.decoder.decoder*), 95
parse_args() (*in module mindaffectBCI.decoder.sigViewer*), 117
parse_args() (*in module mindaffectBCI.online_bci*), 140
perrModeOutput() (*mindaffectBCI.utopia2output.Utopia2Output method*), 146
plos_one() (*in module mindaffectBCI.decoder.datasets*), 92
plot_decoding_curve() (*in module mindaffectBCI.decoder.decodingCurveSupervised*), 98

plot_erp() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 122
plot_factoredmodel() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 122
plot_Fe() (*in module mindaffectBCI.decoder.scoreStimulus*), 116
plot_Fy() (*in module mindaffectBCI.decoder.scoreOutput*), 115
plot_Fycomparsion() (*in module mindaffectBCI.decoder.scoreOutput*), 115
plot_grand_average_spectrum() (*in module mindaffectBCI.decoder.preprocess*), 112
plot_model() (*mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence method*), 103
plot_model_weights() (*in module mindaffectBCI.decoder.model_fitting*), 106
plot_multicca_solution() (*in module mindaffectBCI.decoder.multipleMCCA*), 108
plot_normalizedScores() (*in module mindaffectBCI.decoder.normalizeOutputScores*), 110
plot_outputscore() (*in module mindaffectBCI.decoder.scoreOutput*), 115
plot_raw_preproc_data() (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface method*), 82
plot_subspace() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 122
plot_summary_statistics() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 123
plot_trial() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 123
plot_trial_summary() (*in module mindaffectBCI.decoder.analyse_datasets*), 91
plot_trial_summary() (*in module mindaffectBCI.decoder.decoder*), 95
plotCxy() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 122
pop() (*mindaffectBCI.noisetag.GSM method*), 133
power() (*mindaffectBCI.decoder.UtopiaDataInterface.power_tracker method*), 85
power_tracker (*class in mindaffectBCI.decoder.UtopiaDataInterface*), 84
predict() (*mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence method*), 103
predict_proba() (*mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence method*), 104
PredictedTargetDist (*class in mindaffectBCI.utopiaclient*), 152
PredictedTargetProb (*class in mindaffectBCI.utopiaclient*), 152
PredictionPhase (*class in mindaffectBCI.noisetag*), 137
preprocess() (*in module mindaffectBCI.decoder.preprocess*), 112
preprocess_message() (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface method*), 82
print_decoding_curve() (*in module mindaffectBCI.decoder.decodingCurveSupervised*), 98
processDataPacket() (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface method*), 82
processStimulusEvent() (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface method*), 82
push() (*mindaffectBCI.noisetag.GSM method*), 133
push_back_newmsgs() (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface method*), 82

R

randomSummaryStats() (*in module mindaffectBCI.decoder.utils*), 127
RawMessage (*class in mindaffectBCI.utopiaclient*), 153
read_buffer_offline_data() (*in module mindaffectBCI.decoder.offline.read_buffer_offline*), 75
read_buffer_offline_events() (*in module mindaffectBCI.decoder.offline.read_buffer_offline*), 75
read_buffer_offline_header() (*in module mindaffectBCI.decoder.offline.read_buffer_offline*), 75
read_clientip() (*in module mindaffectBCI.decoder.offline.read_mindaffectBCI*), 77
read_clientts() (*in module mindaffectBCI.decoder.offline.read_mindaffectBCI*), 77
read_DataHeader() (*in module mindaffectBCI.decoder.offline.read_mindaffectBCI*), 76
read_DataPacket() (*in module mindaffectBCI.decoder.offline.read_mindaffectBCI*), 76
read_mindaffectBCI_data_messages() (*in module mindaffectBCI.decoder.offline.read_mindaffectBCI*), 77
read_mindaffectBCI_message()

(*in module mindaffectBCI.decoder.offline.read_mindaffectBCI*), 77
read_mindaffectBCI_messages() (*in module mindaffectBCI.decoder.offline.read_mindaffectBCI*), 77
read_ModeChange() (*in module mindaffectBCI.decoder.offline.read_mindaffectBCI*), 76
read_NewTarget() (*in module mindaffectBCI.decoder.offline.read_mindaffectBCI*), 76
read_recievedts() (*in module mindaffectBCI.decoder.offline.read_mindaffectBCI*), 78
read_Selection() (*in module mindaffectBCI.decoder.offline.read_mindaffectBCI*), 76
read_serverts() (*in module mindaffectBCI.decoder.offline.read_mindaffectBCI*), 78
read_StimulusEvent() (*in module mindaffectBCI.decoder.offline.read_mindaffectBCI*), 76
readArray() (*mindaffectBCI.stimseq.StimSeq static method*), 144
readCapInf() (*in module mindaffectBCI.decoder.readCapInf*), 115
recvall() (*mindaffectBCI.utopiaclient.UtopiaClient method*), 157
redraw_plots() (*in module mindaffectBCI.decoder.decoder*), 95
removeSubscription() (*mindaffectBCI.noisetag.Noisetag method*), 135
removeSubscription() (*mindaffectBCI.utopiaController.UtopiaController method*), 148
Reset (*class in mindaffectBCI.utopiaclient*), 153
reset() (*mindaffectBCI.decoder.lower_bound_tracker.lower_bound_tracker method*), 101
rewrite_timestamps2servertimestamps() (*in module mindaffectBCI.decoder.offline.read_mindaffectBCI*), 78
RingBuffer (*class in mindaffectBCI.decoder.utils*), 126
rmBadChannels() (*in module mindaffectBCI.decoder.preprocess*), 113
rmBadTrial() (*in module mindaffectBCI.decoder.preprocess*), 113
robust_mean() (*in module mindaffectBCI.decoder.utils*), 127
robust_timestamp_regression() (*in module mindaffectBCI.decoder.offline.read_mindaffectBCI*), 78
robust_whitener() (*in module mindaffectBCI.decoder.multipleCCA*), 107
robust_whitener() (*in module mindaffectBCI.decoder.multipleMCCA*), 108
run() (*in module mindaffectBCI.decoder.ccaViewer*), 91
run() (*in module mindaffectBCI.decoder.decoder*), 95
run() (*in module mindaffectBCI.decoder.sigViewer*), 117
run() (*in module mindaffectBCI.decoder.startUtopiaHub*), 117
run() (*in module mindaffectBCI.decoder.trigger_check*), 118
run() (*in module mindaffectBCI.online_bci*), 140
run() (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface method*), 82
run() (*mindaffectBCI.utopia2output.Utopia2Output method*), 146
run_analysis() (*in module mindaffectBCI.decoder.analyse_datasets*), 91

S

save_butter_sosfilt_coeff() (*in module mindaffectBCI.decoder.utils*), 128
score() (*mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence method*), 104
scoreOutput() (*in module mindaffectBCI.decoder.scoreOutput*), 115
scoreStimulus() (*in module mindaffectBCI.decoder.scoreStimulus*), 116
scoreStimulusCont() (*in module mindaffectBCI.decoder.scoreStimulus*), 116
scoreStimulusEpoch() (*in module mindaffectBCI.decoder.scoreStimulus*), 117
scoreStimulusEpoch_factored() (*in module mindaffectBCI.decoder.scoreStimulus*), 117
scoreStimulusEpoch_full() (*in module mindaffectBCI.decoder.scoreStimulus*), 117
search_directories_for_file() (*in module mindaffectBCI.decoder.utils*), 128
Selection (*class in mindaffectBCI.utopiaclient*), 154
selection() (*mindaffectBCI.utopiaController.UtopiaController method*), 148
selectionModeOutput() (*mindaffectBCI.utopia2output.Utopia2Output method*), 146
send_prediction() (*in module mindaffectBCI.decoder.decoder*), 96

sendHeartbeatIfTimeout() (*mindaffectBCI.utopiaclient.UtopiaClient method*), 158
sendMessage() (*mindaffectBCI.decoder.FileProxyHub.FileProxyHub method*), 79
sendMessage() (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface method*), 82
sendMessage() (*mindaffectBCI.utopiaclient.UtopiaClient method*), 158
sendMessages() (*mindaffectBCI.utopiaclient.UtopiaClient method*), 158
sendRaw() (*mindaffectBCI.utopiaclient.UtopiaClient method*), 158
sendRawUDP() (*mindaffectBCI.utopiaclient.UtopiaClient method*), 158
sendStimulusEvent() (*mindaffectBCI.utopiaController.UtopiaController method*), 149
sendStimulusState() (*mindaffectBCI.noisetag.Noisetag method*), 136
serialize() (*mindaffectBCI.utopiaclient.DataHeader method*), 150
serialize() (*mindaffectBCI.utopiaclient.DataPacket method*), 150
serialize() (*mindaffectBCI.utopiaclient.Heartbeat method*), 151
serialize() (*mindaffectBCI.utopiaclient.Log method*), 151
serialize() (*mindaffectBCI.utopiaclient.ModeChange method*), 151
serialize() (*mindaffectBCI.utopiaclient.NewTarget method*), 152
serialize() (*mindaffectBCI.utopiaclient.PredictedTargetDist method*), 152
serialize() (*mindaffectBCI.utopiaclient.PredictedTargetProb method*), 153
serialize() (*mindaffectBCI.utopiaclient.RawMessage method*), 153
serialize() (*mindaffectBCI.utopiaclient.Reset method*), 154
serialize() (*mindaffectBCI.utopiaclient.Selection method*), 154
serialize() (*mindaffectBCI.utopiaclient.SignalQuality method*), 155
serialize() (*mindaffectBCI.utopiaclient.StimulusEvent method*), 155
serialize() (*mindaffectBCI.utopiaclient.Subscribe method*), 156
set_isi() (*mindaffectBCI.noisetag.Noisetag method*), 136
setActiveObjIDs() (*mindaffectBCI.noisetag.Noisetag method*), 136
setnumActiveObjIDs() (*mindaffectBCI.noisetag.Noisetag method*), 136
setStimRate() (*in module mindaffectBCI.stimseq*), 145
setStimRate() (*mindaffectBCI.stimseq.StimSeq method*), 145
setTimeStampClock() (*mindaffectBCI.noisetag.Noisetag method*), 136
setTimeStampClock() (*mindaffectBCI.utopiaclient.UtopiaClient method*), 158
setTimeStampClock() (*mindaffectBCI.utopiaController.UtopiaController method*), 149
shape (*mindaffectBCI.decoder.utils.RingBuffer attribute*), 126
shutdown() (*in module mindaffectBCI.online_bci*), 140
SignalQuality (*class in mindaffectBCI.utopiaclient*), 154
sigViewer() (*in module mindaffectBCI.decoder.sigViewer*), 117
SingleTrial (*class in mindaffectBCI.noisetag*), 137
sklearn_fit() (*mindaffectBCI.decoder.model_fitting.LinearSklearn static method*), 105
sliceData() (*in module mindaffectBCI.decoder.utils*), 128
sliceY() (*in module mindaffectBCI.decoder.utils*), 128
softmax() (*in module mindaffectBCI.decoder.zscore2Ptgt_softmax*), 130
softmax_nout_corr() (*in module mindaffectBCI.decoder.normalizeOutputScores_streamed*), 111
softmax_nout_corr() (*in module mindaffectBCI.decoder.zscore2Ptgt_softmax*), 130
softmax_vs_nout() (*in module mindaffectBCI.decoder.normalizeOutputScores_streamed*), 111
sosfilt_2d_py() (*in module mindaffectBCI.decoder.utils*), 128
sosfilt_zi_py() (*in module mindaffectBCI.decoder.utils*), 128
sosfilt_zi_warmup() (*in module mindaffectBCI.decoder.utils*), 128

spatially_whiten() (*in module mindaffectBCI.decoder.preprocess*), 113
spectrally_whiten() (*in module mindaffectBCI.decoder.preprocess*), 113
split() (*mindaffectBCI.decoder.model_fitting.StratifiedKFold method*), 105
squeeze() (*in module mindaffectBCI.decoder.offline.load_cocktail*), 71
ssdpDiscover (*class in mindaffectBCI.ssdpDiscover*), 142
ssdpDiscover() (*in module mindaffectBCI.utopiaclient*), 159
ssdpgroup (*mindaffectBCI.ssdpDiscover.ssdpDiscover attribute*), 143
ssdpv4group (*mindaffectBCI.ssdpDiscover.ssdpDiscover attribute*), 143
ssdpv6group (*mindaffectBCI.ssdpDiscover.ssdpDiscover attribute*), 143
standardize_channel_power() (*in module mindaffectBCI.decoder.preprocess*), 113
startacquisitionProcess() (*in module mindaffectBCI.online_bci*), 141
startCalibration() (*mindaffectBCI.noisetag.Noisetag method*), 136
startDecoderProcess() (*in module mindaffectBCI.online_bci*), 140
startExpt() (*mindaffectBCI.noisetag.Noisetag method*), 136
startFlicker() (*mindaffectBCI.noisetag.Noisetag method*), 136
startFlickerWithSelection() (*mindaffectBCI.noisetag.Noisetag method*), 137
startHubProcess() (*in module mindaffectBCI.online_bci*), 141
startPrediction() (*mindaffectBCI.noisetag.Noisetag method*), 137
startPresentationProcess() (*in module mindaffectBCI.online_bci*), 141
startSingleTrial() (*mindaffectBCI.noisetag.Noisetag method*), 137
stim2event() (*in module mindaffectBCI.decoder.stim2event*), 118
stim2event() (*mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence method*), 104
stim2eventfilt (*class in mindaffectBCI.decoder.UtopiaDataInterface*), 85
StimSeq (*class in mindaffectBCI.stimseq*), 144
stimSeq (*mindaffectBCI.stimseq.StimSeq attribute*), 145
stimTime_ms (*mindaffectBCI.stimseq.StimSeq attribute*), 145
StimulusEvent (*class in mindaffectBCI.utopiaclient*), 155
StratifiedKFold (*class in mindaffectBCI.decoder.model_fitting*), 105
strip_unused() (*in module mindaffectBCI.decoder.decoder*), 96
Subscribe (*class in mindaffectBCI.utopiaclient*), 155
subscribe() (*mindaffectBCI.noisetag.Noisetag method*), 137
subscribe() (*mindaffectBCI.utopiaController.UtopiaController method*), 149
sumstats (*class in mindaffectBCI.noisetag*), 139

T

tactile_PatientStudy() (*in module mindaffectBCI.decoder.datasets*), 92
tactileP3() (*in module mindaffectBCI.decoder.datasets*), 92
temporal_decorrelator (*class in mindaffectBCI.decoder.UtopiaDataInterface*), 86
temporally_decorrelate() (*in module mindaffectBCI.decoder.preprocess*), 114
terminate() (*mindaffectBCI.online_bci.NoneProc method*), 139
test_butter_sosfilt() (*in module mindaffectBCI.decoder.utils*), 128
test_compCxx_diag() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 124
test_compCyx_diag() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 124
test_fir() (*in module mindaffectBCI.decoder.preprocess*), 114
test_loader() (*in module mindaffectBCI.decoder.datasets*), 92
test_sosfilt_py() (*in module mindaffectBCI.decoder.utils*), 128
testcase() (*in module mindaffectBCI.decoder.datasets*), 92
testcase() (*in module mindaffectBCI.decoder.decodingCurveSupervised*), 98
testcase() (*in module mindaffectBCI.decoder.decodingSupervised*), 100
testcase() (*in module mindaffectBCI.decoder.FileProxyHub*), 79
testcase() (*in module mindaffectBCI.decoder.model_fitting*), 106
testcase() (*in module mindaffectBCI.decoder.multipleCCA*), 107
testcase() (*in module mindaffectBCI.decoder.multipleMCCA*), 108

testcase() (*in module mindaffectBCI.decoder.normalizeOutputScores*), 110
testcase() (*in module mindaffectBCI.decoder.normalizeOutputScores_streamed*), 111
testcase() (*in module mindaffectBCI.decoder.offline.load_brainsonfire*), 70
testcase() (*in module mindaffectBCI.decoder.offline.load_brainstream*), 70
testcase() (*in module mindaffectBCI.decoder.offline.load_cocktail*), 71
testcase() (*in module mindaffectBCI.decoder.offline.load_mark_EMG*), 71
testcase() (*in module mindaffectBCI.decoder.offline.load_mindaffectBCI*), 73
testcase() (*in module mindaffectBCI.decoder.offline.load_ninapro_db2*), 73
testcase() (*in module mindaffectBCI.decoder.offline.load_openBMI*), 74
testcase() (*in module mindaffectBCI.decoder.offline.load_p300_prn*), 75
testcase() (*in module mindaffectBCI.decoder.offline.load_twofinger*), 75
testcase() (*in module mindaffectBCI.decoder.offline.read_buffer_offline*), 75
testcase() (*in module mindaffectBCI.decoder.offline.read_mindaffectBCI*), 78
testCase() (*in module mindaffectBCI.decoder.readCapInf*), 115
testcase() (*in module mindaffectBCI.decoder.scoreStimulus*), 117
testcase() (*in module mindaffectBCI.decoder.stim2event*), 118
testcase() (*in module mindaffectBCI.decoder.zscore2Ptgt_softmax*), 130
testcase() (*mindaffectBCI.decoder.lower_bound_tracker.lower_bound_tracker static method*), 101
testcase() (*mindaffectBCI.decoder.UtopiaDataInterface.butterfilt_and_downsample static method*), 84
testcase() (*mindaffectBCI.decoder.UtopiaDataInterface.channel_power_standardize method*), 84
testcase() (*mindaffectBCI.decoder.UtopiaDataInterface.power_tracker static method*), 85
testcase() (*mindaffectBCI.decoder.UtopiaDataInterface.stim2eventfilt method*), 85
testcase() (*mindaffectBCI.decoder.UtopiaDataInterface.temporal_decorrelator method*), 86
testcase() (*mindaffectBCI.decoder.UtopiaDataInterface.timestamp_interpolation method*), 87
testCase_filterbank() (*in module mindaffectBCI.decoder.preprocess*), 114
testcase_matlab_summarystatistics() (*in module mindaffectBCI.decoder.multipleCCA*), 107
testCase_spectralwhiten() (*in module mindaffectBCI.decoder.preprocess*), 114
testCase_temporallydecorrelate() (*in module mindaffectBCI.decoder.preprocess*), 114
testcases() (*in module mindaffectBCI.decoder.scoreOutput*), 116
testCases() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 124
testComputationMethods() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 124
testCyy2() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 124
testdataset() (*in module mindaffectBCI.decoder.datasets*), 92
testElectrodeQualities() (*in module mindaffectBCI.decoder.UtopiaDataInterface*), 86
testERP() (*in module mindaffectBCI.decoder.UtopiaDataInterface*), 86
testFileProxy() (*in module mindaffectBCI.decoder.UtopiaDataInterface*), 86
testFileProxy2() (*in module mindaffectBCI.decoder.UtopiaDataInterface*), 87
testIncrementalScanning() (*in module mindaffectBCI.ssdpDiscover*), 143
testLeadLag() (*in module mindaffectBCI.decoder.model_fitting*), 106
testNoSignal() (*in module mindaffectBCI.decoder.utils*), 128
testPP() (*in module mindaffectBCI.decoder.UtopiaDataInterface*), 87
testRaw() (*in module mindaffectBCI.decoder.UtopiaDataInterface*), 87
testSending() (*in module mindaffectBCI.utopiaclient*), 159
testStandardization() (*in module mindaffectBCI.utopiaclient*), 159
testSignal() (*in module mindaffectBCI.decoder.utils*), 128
testSlicedvsContinuous() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 124
testtestSignal() (*in module mindaffectBCI.decoder.utils*), 128

timestamp_interpolation (*class in mindaffectBCI.decoder.UtopiaDataInterface*), 87
TimeStampClock (*class in mindaffectBCI.utopiaclient*), 156
timestampPlot() (*in module mindaffectBCI.decoder.timestamp_check*), 118
toFile() (*mindaffectBCI.stimseq.StimSeq method*), 145
toy() (*in module mindaffectBCI.decoder.datasets*), 92
transform() (*mindaffectBCI.decoder.lower_bound_tracker.lower_bound_tracker method*), 102
transform() (*mindaffectBCI.decoder.model_fitting.BaseSequence2Sequence method*), 104
transform() (*mindaffectBCI.decoder.UtopiaDataInterface.butterfilt_and_downsample method*), 84
transform() (*mindaffectBCI.decoder.UtopiaDataInterface.channel_power_standardize method*), 84
transform() (*mindaffectBCI.decoder.UtopiaDataInterface.power_tracker method*), 85
transform() (*mindaffectBCI.decoder.UtopiaDataInterface.stim2eventfilt method*), 85
transform() (*mindaffectBCI.decoder.UtopiaDataInterface.temporal_decorrelator method*), 86
transform() (*mindaffectBCI.decoder.UtopiaDataInterface.timestamp_interpolation method*), 87
transform() (*mindaffectBCI.decoder.UtopiaDataInterface.TransformerMixin method*), 79
TransformerMixin (*class in mindaffectBCI.decoder.UtopiaDataInterface*), 79
transpose() (*in module mindaffectBCI.stimseq*), 145
trigger_check() (*in module mindaffectBCI.decoder.trigger_check*), 118
triggerPlot() (*in module mindaffectBCI.decoder.trigger_check*), 118
try_connect() (*mindaffectBCI.utopiaclient.UtopiaClient method*), 158
twofinger() (*in module mindaffectBCI.decoder.datasets*), 92

U

unwrap() (*in module mindaffectBCI.decoder.utils*), 129
unwrap() (*mindaffectBCI.decoder.utils.RingBuffer method*), 126
unwrap_test() (*in module mindaffectBCI.decoder.utils*), 129
update() (*mindaffectBCI.decoder.lower_bound_tracker.lower_bound_tracker method*), 102
update() (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface method*), 82
update_and_send_ElectrodeQualities() (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface method*), 83
update_codebook_permutation() (*mindaffectBCI.noisetag.Flicker method*), 132
update_electrode_powers() (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface method*), 83
update_is() (*mindaffectBCI.noisetag.Flicker method*), 132
update_statistics() (*mindaffectBCI.noisetag.sumstats method*), 139
updateCxx() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 124
updateCxy() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 124
updateCyy() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 124
updateStimulusState() (*mindaffectBCI.noisetag.Noisetag method*), 137
updateSummaryStatistics() (*in module mindaffectBCI.decoder.updateSummaryStatistics*), 125
upsample_codebook() (*in module mindaffectBCI.decoder.utils*), 129
upsample_stimseq() (*in module mindaffectBCI.decoder.devent2stimsequence*), 101
Utopia2Output (*class in mindaffectBCI.utopia2output*), 146
UTOPIA_SSDP_SERVICE (*mindaffectBCI.utopiaclient.UtopiaClient attribute*), 156
UtopiaClient (*class in mindaffectBCI.utopiaclient*), 156
UtopiaController (*class in mindaffectBCI.utopiaController*), 147
UtopiaDataInterface (*class in mindaffectBCI.decoder.UtopiaDataInterface*), 79
UtopiaMessage (*class in mindaffectBCI.utopiaclient*), 158

V

VERBOSITY (*mindaffectBCI.decoder.UtopiaDataInterface.UtopiaDataInterface attribute*), 80

visPtgt() (*in module mindaffectBCI.decoder.zscore2Ptgt_softmax*), 130
visualize_Fy_Py() (*in module mindaffectBCI.decoder.model_fitting*), 106

W

wait_msearch_response() (*mindaffectBCI.ssdpDiscover.ssdpDiscover method*), 143
WaitFor (*class in mindaffectBCI.noisetag*), 138
window_axis() (*in module mindaffectBCI.decoder.utils*), 129

X

xy2latlong() (*in module mindaffectBCI.decoder.readCapInf*), 115
xyz2xy() (*in module mindaffectBCI.decoder.readCapInf*), 115

Z

zero_outliers() (*in module mindaffectBCI.decoder.utils*), 129
zscore2Ptgt_softmax() (*in module mindaffectBCI.decoder.zscore2Ptgt_softmax*), 130