MindAffect BCI keyboard: System documentation

Contents

1 General purpose **2**

2 Design choices **2**
- 2.1 Requirements . . . 2
  - 2.1.1 Graphical Framework . . . 2
  - 2.1.2 Multiple keyboards . . . 2
  - 2.1.3 Keyboard extensions . . . 2
- 2.2 Code Structure . . . 3
  - 2.2.1 Framework Facade . . . 3
  - 2.2.2 Settings Manager . . . 3
  - 2.2.3 Keyboard extensions . . . 4
  - 2.2.4 create_ngram_file . . . 4
  - 2.2.5 Window classes . . . 4
  - 2.2.6 Key, KeyType, and FunctionalProvider . . . 4

3 System files **5**
- 3.1 Keyboard layouts . . . 5
- 3.2 Configuration files . . . 5
- 3.3 Frequency lists . . . 5
- 3.4 N-gram files . . . 6
- 3.5 Exceptions log . . . 6

4 System overview **7**

5 System flow **8**

6 Peripheral interactions **10**
- 6.1 Utopia Hub . . . 10

1 General purpose

The keyboard is designed to interact with the pymindaffectBCI of Mindaffect and has many supporting features. It uses the Psychopy framework to run the GUI. It can follow any layout. By default it uses the qwerty layout with some additional buttons. The keyboard can be calibrated in the application before the first use. Afterwards the user will be able to type using the BCI. The keyboard supports the typing by having word correction and prediction in many different languages. These can be disabled in the settings menu. Additionally, the keyboard has a Text-to-speech (TTS) functionality to speak the written text aloud. This is also available in multiple languages. For additional accessibility the text size can be changed in the settings menu. All settings are saved for the next use. Finally, the text can be saved to a text file.

Figure 1: A keyboard with the qwerty-layout

2 Design choices

2.1 Requirements

2.1.1 Graphical Framework

One of the most fundamental parts of this application is the graphical framework. The timing accuracy of the key flickering gravely influences the performance of the keyboard and thus the ability to type with the EEG-headset, which is a hard requirement. To decide on which graphical framework to use, we ran tests on several frameworks to see which was the most accurate and consistent. The two most promising frameworks were Psychopy and Pyglet. We decided to implement the keyboard using the former.

2.1.2 Multiple keyboards

Another requirement was to have multiple configurable keyboards, for example keyboards with lower case letters, upper case letters, numerical characters, special characters, etc. This was realized in the shape of a keyboard containing multiple keypads (sub-keyboards). Each keypad is configurable by adding and/or changing a json file in the 'keypad_layouts' folder. The keypads have been programmed in such a way that special function keys can be used to switch between keypads.

2.1.3 Keyboard extensions

Three of the requirements were text-to-speech, predictive text, and word correction.

The text-to-speech functionality is linked to a button with an audio wave icon, as can be seen in the lower left corner of Figure 1. The last line that has been typed is read out loud. This way, the text does not have to be completely erased to avoid repeating typed text. For text-to-speech,

three different modes can be used; paid_online, free_online, and offline. As the names suggest, both paid_online and free_online are online text-to-speech services. The paid service generates higher-quality speech than the free service. Both of the online services generate higher-quality speech than the offline version. Hence, the offline version is mostly used as a backup for when there is no internet connection available.

Both predictive text and word correction have been integrated into the keyboard using 'prediction keys' as we call them. These are the keys displayed above the general character keys. While typing, these keys show suggestions for word correction, word completion, and next word prediction, depending on what the user is typing (correction and completion when the user typed part of a word, next word prediction when a new word is not started yet).

Initially, prediction and completion were implemented using a GPT2 network. While this network generally supplied good (context-dependent) suggestions, it required downloading several gigabytes of model weights and took very long to load, significantly increasing the application start-up time. In addition, this model only supported English. Instead, we decided to implement an n-gram frequency model. Even though the suggestions are less context-dependent, the method is a lot more light-weight, supports multiple languages, and even learns when the user is typing using the keyboard. Three languages (English, German, and Dutch) are pre-trained by us for the model. If developers wish to add langauges, this can easily be done by feeding (large) plain text files to the 'create_ngram_file' program we have written.

The word correction module has been implemented based on frequency lists. If a user has an internet connection, the right language frequency list is downloaded automatically. If not, the 'create_ngram_file's -dictionary option can generate such a frequency list for a language.

2.2 Code Structure

During the development of this project, a lot of structural design choices had to be made. In general, we tried to make sure that the code for the keyboard was as modular as possible in order to make maintenance more doable and allow for easier modification or even replacement of the features. On top of that, we used a few design patterns to clean up the code and/or simplify a problem.

2.2.1 Framework Facade

One of the most prominent design patterns used, is the facade for the graphical frameworks. The framework we currently use is Psychopy, which has some odd implementations and functionality. To make it easier to work with Psychopy, we created a 'Framework Facade' to make only the functionality we need easily usable and to hide the rest.

Given the importance of timing of the key flickering, the graphical framework gravely influences the performance and accuracy of the keyboard. Because of this, it may be desirable to change the graphical framework to, for example, run tests with other frameworks. To make it easier to switch frameworks, we implemented the Framework Facade as an 'abstract class' (as far that is possible in Python). For each new framework, a new facade class can be extended from this general facade class (in our case, this is the Psychopy Facade class).

2.2.2 Settings Manager

The Settings Manager plays the most important role in the configurability of the keyboard for the users. If any user settings change, these changes have to be forwarded to a lot of components throughout the keyboard. For example, if a user changes the keyboard language, the Text-to-speech, Word Correction, and Word Prediction modules all have to be (partially) re-initialized.

To do this in a neat way, we used an observer pattern. The modules subscribe to the Settings Manager and get notified if any user settings change. Not only does this simplify the code for updating the modules, it also makes it easier to add more modules that depend on these settings.

The Settings Manager also interacts with dictionary_retrieval.py to ensure that the right frequency lists are downloaded whenever a language is switched.

2.2.3 Keyboard extensions

The features that added extra functionality to the keyboard, i.e. Text-to-speech, Word Correction, and Word Prediction, could easily be made modular. They all work with a few functions that take (usually text) input and produce the required output, all without interacting too much with other modules within the project.

Text-to-speech
The Text-to-speech module is an interface between three possible text-to-speech services. It could also be considered a small facade for these services. If specified, the module uses an online speech generation service, unless no internet connection is available. If the latter is the case or if specified, an offline speech generation service is used. This means that there is always a backup in case a connection is not available or not working. Within the project, Text-to-speech only directly relies on the Exception Logger for (a part of) its functionality.

Word Correction
The Word Correction module is a small facade for the Symspellpy library. It simply creates a SymSpell object and calls the appropriate functions whenever the `correct()` function is called. Within the project, Word Correction does not directly rely on any other classes for its functionality.

Word Prediction
The Word Prediction module is a larger module compared to the Text-to-speech and Word Correction modules. This module does not rely on any external libraries, because everything has been implemented by our team to completely fit our use-case. Within the project, Word Prediction does not directly rely on any other classes for its functionality.

2.2.4 create_ngram_file

In order to be able to use the Word Prediction module, an n-gram dictionary has to be generated. This can be done either by the Word Prediction module itself, which is done based on the user-typed text, or by processing text corpora.

The latter is done by the create_ngram_file script. This script is a small stand-alone program that has no interactions with the keyboard. We implemented it separately because it is more part of the set-up process than actual interaction with the keyboard. A general user would not need to use this program if the desired language has already been added. Also, the way the program is implemented, it can be easily reused for purposes other than our keyboard's prediction module.

2.2.5 Window classes

In our application, a Window is any screen that is displayed. There are a lot of different kinds of Windows, like a MenuWindow, KeyboardWindow, or a CalibrationWindow. They have a shared abstract Window class they extend from. When navigating through the menu and keypads, different Windows have to be activated and deactivated.

Initially the application kept track of a list of Windows and had a variable to track the Window that was currently active. Even though this was a relatively easy way to do it, it led to some hard-coding and made it more tedious to add new Windows. Now, the Windows are set up in a recursive structure, so whenever a Window has to be switched, the Window to switch to can easily be found using a search through the recursive tree. This saves a lot of bookkeeping, while not affecting the performance of the application.

2.2.6 Key, KeyType, and FunctionalProvider

Within the keyboard, a lot of Keys are used. Some of the Keys have the basic functionality of typing the character they represent, but others have special functions like the Backspace Key, Text-to-speech Key, or the Menu Key.

Initially, we made a general Key class which each different Key extended and overwrote functions from. However, with the large amount of special Keys that only very slightly modified the general Key's behavior it quickly resulted in a large amount of duplicate code in the form of subclass definitions. To simplify this part of the code, we instead created a general Key class with the basic Key functionality. Additionally, each Key class instance receives a KeyType (enumeration)

value, which indicates what kind of behavior the Key should have. The special implementations for the different Key functions are supplied by the FunctionalProvider class.

3 System files

The system makes use of pre-made keyboard layouts, config files, frequency lists and n-gram files.

3.1 Keyboard layouts

For every keyboard a different layout exists. These are all stored in the folder named "keypad_layouts". The layouts are stored in a JSON format as can be seen in the figure below. In this case it shows the full code for the lower case layout. The files are most often loaded using the load_json function in KeyboardLoader.py.

```
[
  [
    ["BASIC_KEY","q"], ["BASIC_KEY","w"], ["BASIC_KEY","e"], ["BASIC_KEY","r"], ["
    BASIC_KEY","t"],
    ["BASIC_KEY","y"], ["BASIC_KEY","u"], ["BASIC_KEY","i"], ["BASIC_KEY","o"], ["
    BASIC_KEY","p"]
  ],
  [
    ["BASIC_KEY","a"], ["BASIC_KEY","s"], ["BASIC_KEY","d"], ["BASIC_KEY","f"], ["
    BASIC_KEY","g"],
    ["BASIC_KEY","h"], ["BASIC_KEY","j"], ["BASIC_KEY","k"], ["BASIC_KEY","l"]
  ],
  [
    ["BASIC_KEY","z"], ["BASIC_KEY","x"], ["BASIC_KEY","c"], ["BASIC_KEY","v"], ["
    BASIC_KEY","b"],
    ["BASIC_KEY","n"], ["BASIC_KEY","m"], ["CLEAR_KEY","key_icons\\clear_default.
    png"],
    ["BACKSPACE_KEY","key_icons\\delete_space_default.png"]
  ],
  [
    ["SWITCH_TO_MENU_KEY","key_icons\\settings_default.png"], ["TTS_KEY","key_icons
    \\speech_default.png"],
    ["SWITCH_TO_SYMBOLS_KEY","?123"], ["SWITCH_TO_UPPER_KEY","ABC"], ["BASIC_KEY","
    ,"], ["SPACE_BAR_KEY","Space"],
    ["BASIC_KEY","."], ["ENTER_KEY","key_icons\\enter_default.png"]
  ]
]
```

3.2 Configuration files

There are two different configuration files which can be found in the folder named "configs". The file 'user_config.json' contains the settings the user has set as well as standard settings. On top of that it contains settings that can only be tweaked in the file itself. Next to this there is the 'keyboard_config.json'. This different settings depending on which window is active.

3.3 Frequency lists

In the directory dictionaries\frequency_lists the frequency lists for the different languages can be found. These are downloaded from a GitHub page on request of the program using the download function in 'dictionary_retrieval.py'. These lists contain words with an integer denoting their frequency. For example, the first five lines of the Dutch frequency list 'NL.txt' look as follows:

ik 10006772
je 9001167
het 7081845
de 6792979
dat 6007618

3.4 N-gram files

In the directory dictionaries\n-gram_files the n-gram files for the different languages can be found. The files are split in pre-trained and user trained files. Pre-trained files come with the program for English, Dutch and German. They are trained on texts from the Gutenberg project. New n-gram files can be created using create_ngram_file.py on text files in the preferred languages. Additionally, n-gram files are created based on the user's typing when using the program. The n-gram files are json files with dictionaries which track the n-gram frequencies. The dictionary contains a max-depth variable, which indicates the maximum n-gram size used in the dictionary, and for each context entry there is a counter, which indicates how many times that context appeared in the processed text. An example for the sentence "Fred is cool" can be seen below:

```
{
    "__max-depth__": 3,
    "": {"__counter__": 3, "Fred": 1, "is": 1, "cool": 1},
    "Fred": {"__counter__": 1, "is": 1},
    "is": {"__counter__": 1, "cool": 1},
    "Fred is": {"__counter__": 1, "cool": 1}
}
```

3.5 Exceptions log

During every run of the program, an exceptions file 'exceptions.log' is created. This keeps track of any exceptions that were thrown during run time of the program. It can be used as a debugging tool to solve any issues that occurred.

4 System overview

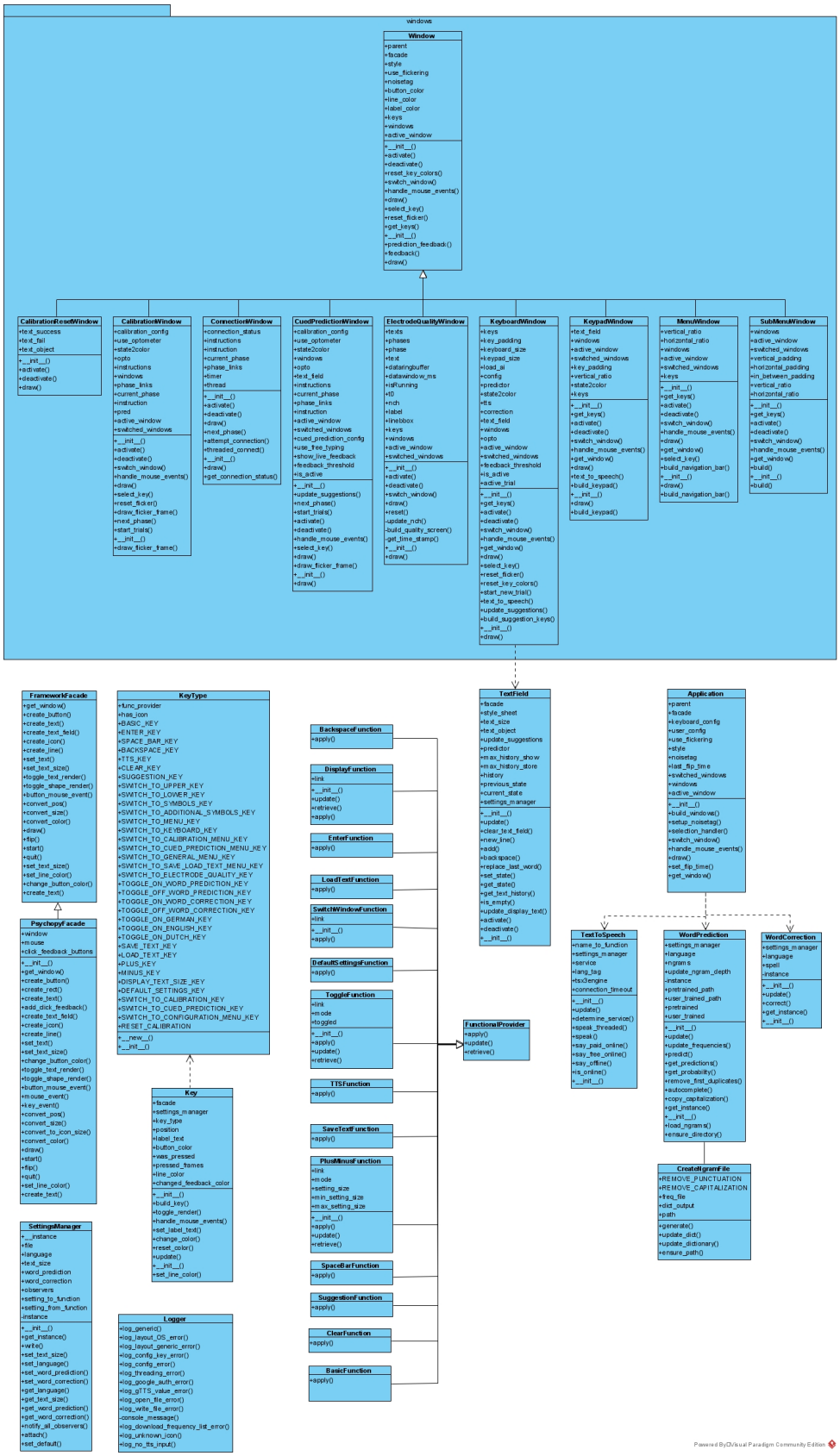

Figure 2: UML diagram of the system

5 System flow

Viewing the system at a high abstraction level it consists of three main processes:

- Instantiation of the graphical elements provided a GUI and instantiation of the external functional modules word correction, word prediction, text-to-speech and the internal functional module settings manager which is used to distribute information within the application. Furthermore, an instance of noisetag is created.
- Drawing the active (to be displayed) graphical elements.
- Handling user input which can either come from the BCI or a mouse.

Note that the latter two exchange information until termination thus making up the event-loop of the application.

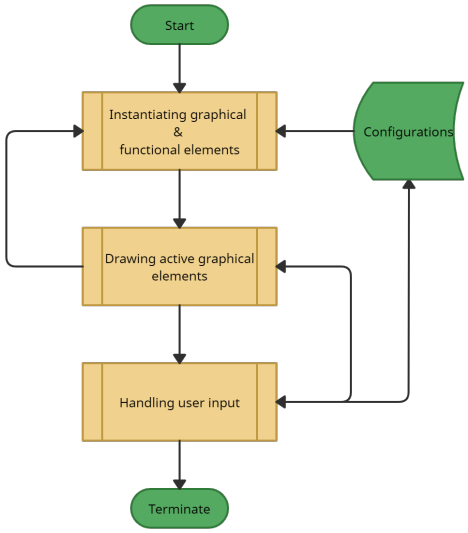

Figure 3: High-level flow chart of the system

On a narrower abstraction level those three processes can be broken down as follows:

- The first process now includes the order in which the application runs through the instantiation of graphical and functional elements as well as the order in which the former are drawn (without user input).
- The second process now contains the broader interactions between drawing and input handling. Although the user input is not shown explicitly it is contained within the multiple handling processes. Depending on whether noisetag is actually used this might be either input from noisetag itself, the physical keyboard or the mouse.
- The third process now explicitly shows accessibility relations between windows as well as minor degree of data usage and flow.

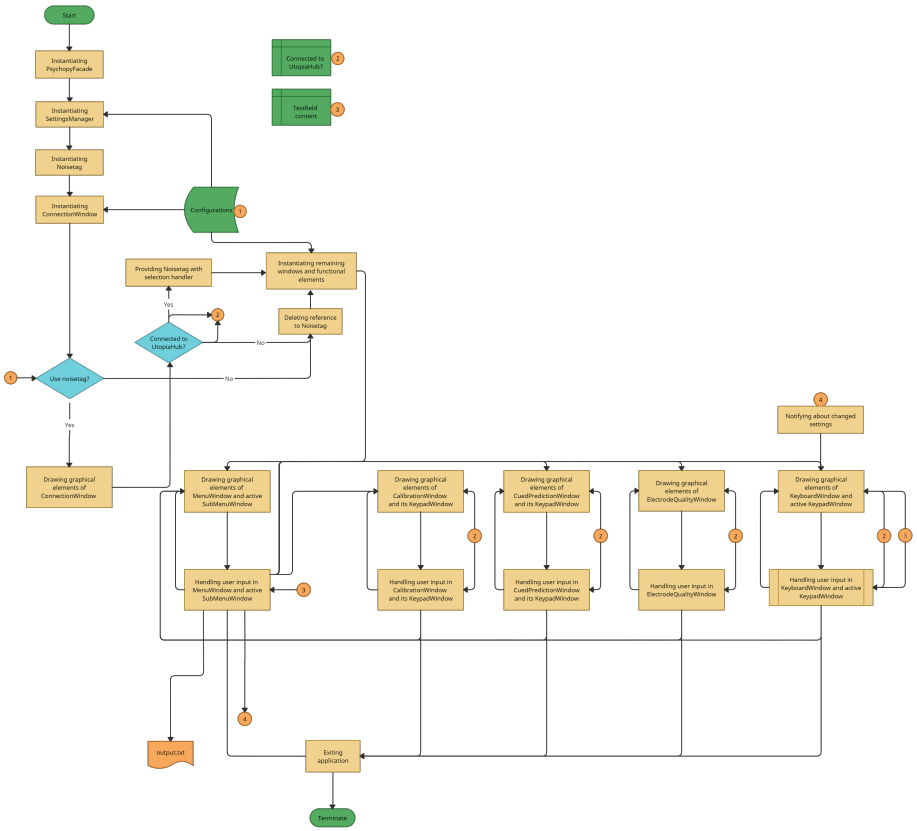

Figure 4: Medium-level flow chart of the system

For the sake of purpose we chose to not break down the displayed processes any further. Yet, because the process of handling user input within the KeyboardWindow is especially important it exemplifies how the system would look like on a low abstraction level.

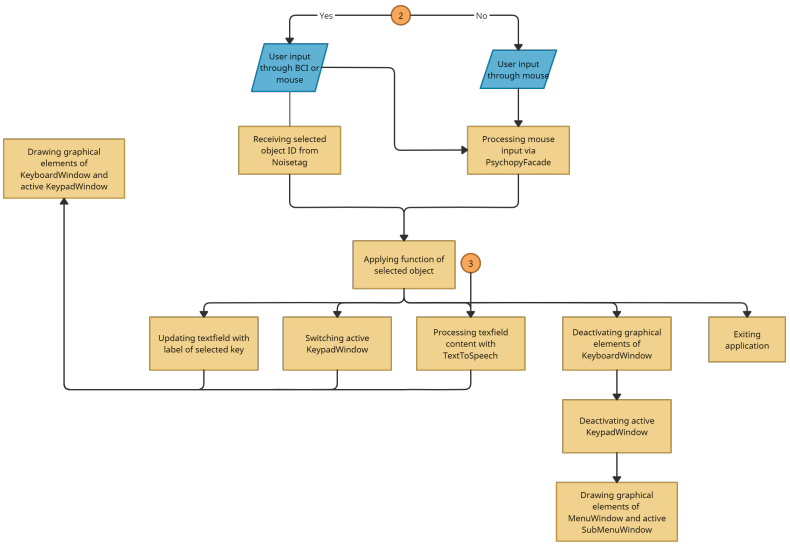

Figure 5: Low-level flow chart of input handling within KeyboardWindow

6 Peripheral interactions

The MindAffect BCI (and thus the keyboard as well) has a set of main physical components, illustrated below:

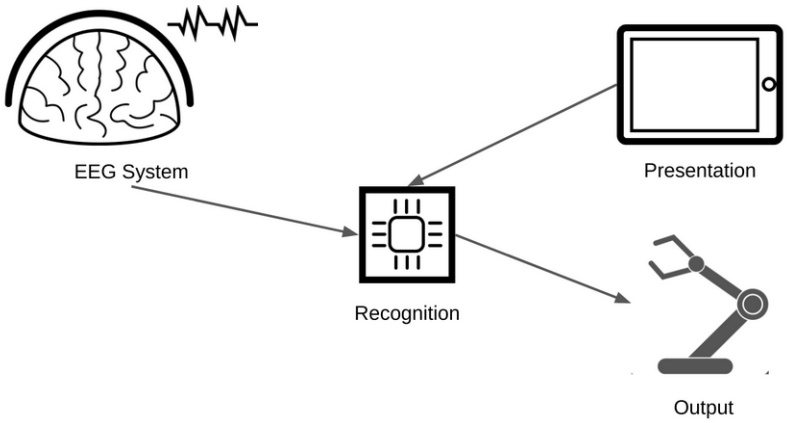

Figure 6: The main physical components of the BCI.

- **Presentation**
  This component is usually something like a screen or display. Its purpose is to show the keyboard and to let the keys flicker.
- **EEG system**
  This component is necessary to pick up the EEG signals (usually via a headset). These signals are then sent to the Decoder to that it can identify the users target key.
- **Recogniser or Decoder**
  This is the physical host of the software that runs the machine learning algorithms and combines the information from the Presentation component with the EEG signals from the EEG system to recognise the users intended target key. It then selects the intended target and forwards this to the Output component. This component is usually a computer like a laptop or a raspberry pi.
- **Output**
  In this case, the output is that the intended target key is selected on the keyboard and that the action that corresponds to this key is performed (type a letter, backspace, etc.).

6.1 Utopia Hub

The Utopia Hub is responsible for connecting together all the other components of the system. It is a purely software component, running on the same platform as the Decoder (on the computer).

It forwards the incoming messages from the Presentation component to the Decoder component and forwards the messages from the Decoder component to the Output component.