SOW-MKI46: Automatic Error-Correction Using Feedback-Related Negativity

Patrick Ebel
p.ebel@student.ru.nl
4399803

Casper van Elteren
c.vanElteren@student.ru.nl
4613600

Lisa Goerke
l.goerke@student.ru.nl
4742869

February 2, 2017

Contents

1 Introduction

Brain Computer Interfacing (BCI) is a method that allows its user to communicate information about her or his mental state without the use of the peripheral nervous system (Van Gerven et al., 2009). Non-invasive transportable neuroimaging devices, such as EEG systems, allow for practical BCI application but suffer from poor signal-to-noise ratio, which results in common misclassifications of the user's intentions. This issue is of outstanding importance, as misclassifications come at the high cost of making the user's mental efforts being in vain, and the associated consequences of misinterpreting them. The problem is made only more severe in cases where the user's latent mental state is not binary-valued, but may take more than two discrete class values.

Here, we propose a solution to the problem of erroneous classifications in BCI, automated error correction. Automated error correction (Parra, Spence, Gerson, & Sajda, 2003) deals with correcting wrongly generated computer commands without requiring any active efforts of the user. A neuronal signature of error, and proxy of misinterpreted BCI commands, are the class of Error-related signals (ErrS) (Chavarriaga, Sobolewski, & Millan, 2014). Importantly, ErrS are not only evoked in response to the BCI user acting incorrectly but also if he or she observes a computer-generated command not matching her or his intentions (Pierre W. Ferrez, 2008) or receiving negative feedback (Ward et al., 2013).

In this report, we make use of ErrS signatures to generate and auto-correct brain-based commands for the Cybathlon BrainRunners game (Cybathlon, 2016). In BrainRunners, a BCI player's avatar runs along a 1D parcours composed of different platform types. The aim of the game is to reach the goal platform as quick as possible, which can be facilitated by generating the correct movement on the corresponding platforms. However, BrainRunners' in-game mechanics also harshly punishes incorrect commands by slowing down the progress of the player's avatar.

In addition to mental state decoding being a naturally challenging problem, the BrainRunners BCI has to decode a total of four classes of imagined movement, occurring at an unknown point in time. In comparison, an Errs signature is binary and shows up within a known time interval, namely briefly after a possibly incorrect command has been generated in-game. This should make ErrS detection a fundamentally easier problem to solve than imagined movement decoding—setting the expectation that ErrS can be useful in correcting a primary movement classifier.

In this report, we investigate whether ErrS are helpful in improving decoding of imagined movements. We show that by applying a BCI composed of two classifiers, one for decoding actively imagined movements and another for detecting passive dependent ErrS, one can achieve better performance in the Cybathlon BrainRunners game: Subjects participating in our study needed less time to complete the BrainRunners parcours due to a reduction of time spent in slow-down mode. Furthermore, participants reported a greater sense of control over their avatar. These results indicate that a multimodal hybrid BCI, decoding both actively imagined movements and passively generated error signals, can make mental state decoding more robust and fun.

2 Methods

2.1 The BCI System

Brain Computer Interfacing (BCI) is a method that allows its user to communicate information about her or his mental state without the use of the peripheral nervous system (Van Gerven et al., 2009). For this study, the following material was used to built an EEG-based BCI system:

Hardware material

- Lenovo ThinkPad T440P (Intel®Core^{TM} i5-4300M CPU @ 2.60GHz $\times$ 4, Intel®Haswell Mobile, Ubuntu 16.04)
- TMSi Mobita 10-channel water-based EEG system
- TMSi 32-channel amplifier, 250 Hz sampling rate

Software material

- Matlab R2015a & Python 2.7
- Buffer BCI (Farquhar, 2016), FieldTrip (Oostenveld, Fries, Maris, & Schoffelen, 2011)
- Cybathlon Brain Runners (Cybathlon, 2016)
- Python packages: numpy, scipy, sklearn, matplotlib, seaborn, h5py

Participants A total of 3 participants were recruited for this study, 1 female and 2 male students. 2 subject were left-handed and the mean age was 25. All subjects reported having previous experience with BCI. Each subject agreed on the Donders Centre for Cognition's consent form and was treated according to the Declaration of Helsinki. Furthermore, each participant received brief instructions on the tasks as outlined in sections 2.2.2 and 2.2.4.

2.2 Experimental Design

Fig. 1 provides a schematic flowchart outlining the experimental design of this study. The preparation phase takes 10 minutes and is done as outlined in section 2.2.1. The calibration block is conducted as described in section 2.2.2 and takes about 15 minutes. Classifier training is done as reported in sections 2.2.1 for about 2 minutes. Finally, two times three runs of testing (with and without decoding ErrS) are conducted as described in section 2.2.4. Each run lasts between 150 and 250 seconds resulting in a total of approximately 18 minutes. All in all, the experiment is designed to take about 45 minutes.

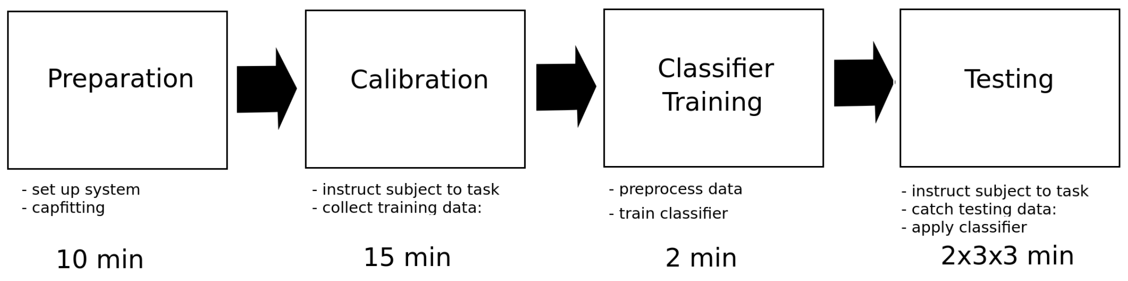


Figure 1: Schematic flow-chart overview of the experiment and its phases of preparation, training, calibration and testing.

2.2.1 Preparation Phase

In the preparation phase, the subject is warmly welcomed and introduced to the terms of the experiment. Once the participant is informed and agreed to the conditions, he or she is situated in front of a computer monitor in a quiet room and the EEG cap is fitted on the head as outlined in the Mobita manual document (Farquhar, 2015). Finally, the Buffer BCI architecture is set up and all necessary scripts are run as described in the user guide of section 5.1 to begin with the calibration phase as stated below. The preparation phase is expected to take about 10 minutes.

2.2.2 Calibration Phase

An important aspect of a successful BCI is to have an experimental design which generates a signal salient enough to be well detectable and decodable on a single-trial basis. Our calibration phase is designed to collect actively imagined movement signals and error-related signals (see section 2.3.1) in a cued imagined movement task with imitated feedback provided to the naive subject. At the beginning of the phase, each participant received the following standardized instructions:

"Dear ⟨ Name ⟩, thanks for participating in our experiment! Please focus on the center of the screen while you'll complete a set of trials. In each trial you have to vividly imagine yourself moving either your left hand, right hand or feet—while cued to do so. At the end of each trial, you receive either positive (green) or negative (red) feedback. Once having received enough positive feedback you can proceed to the next phase."

In detail, the calibration phase consists of a welcome screen followed by 4 sequences of 15 imagined movement epochs each, interleaved with pauses lasting until the participant's indicates that he or she wants to continue. Every epoch consists of a 1.5 sec baseline condition, a 2 sec visually cued movement condition and a 1 sec visual mock feedback stimulus that is either positive or negative, suggesting a successful or an erroneous trial. All three imagined movement types are presented randomly and for a total of 20 trials each. The amount positive and negative feedback are provided is distributed uniformly over the different imagined movement conditions, with a total of 40 positive feedback trials and 20 negative feedback trials.

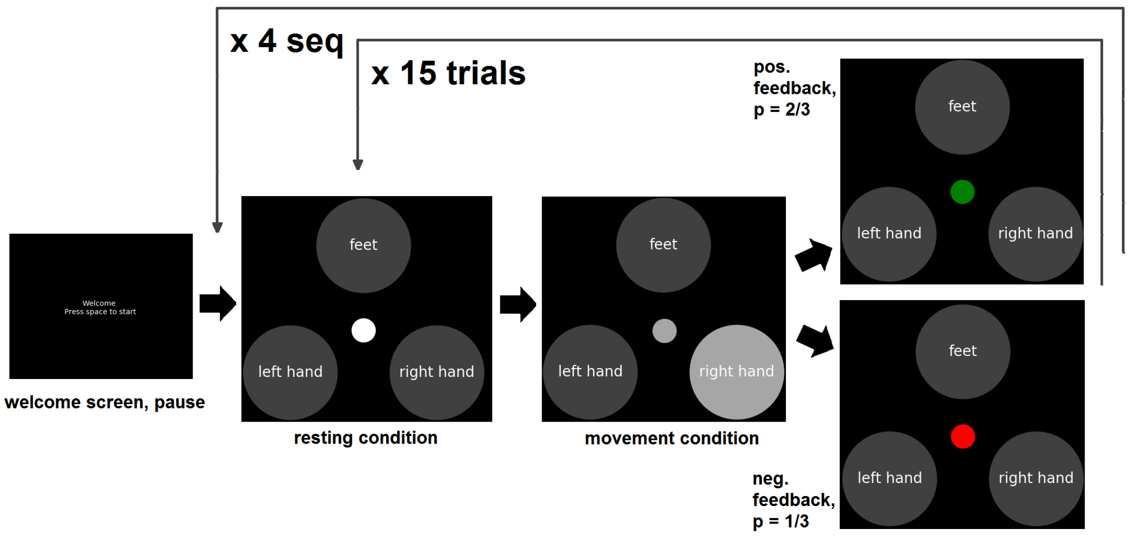


Figure 2: Schematic flow-chart of the calibration phase. A single epoch consists of (i) a resting condition, (ii) a movement condition and (iii) a feedback phase, with a short pause following every fifteenth trial.

The subject was instructed to vividly imagine performing the motor commands as such kinesthetic movement imagination is reported to produce the most salient single-trial detectable signals (Blokland et al., 2015). Moreover, the fixation point displayed at the center of the screen wriggled slightly around its initial position during the imagined movement condition—which served the purpose of mimicking the rich and dynamic stimulus environment to be faced in the testing phase, thus making the recorded brain signals more like those to be encountered in-game. Finally, we made the design choice to provide more positive than negative feedback for two reasons: First, to keep the subject motivated and avoid frustration. Second, to avoid de-sensitization in response to too frequent negative feedback, which might attenuate the evoked error signal. Yet, the collected 20 trials of negative feedback are on pair with the amount of epochs per movement condition and should provide sufficient data to train the classifier on as outlined in section 2.4.1. In sum, the calibration phase requires 15 minutes of time.

2.2.3 Training Phase

The training phase consists of preprocessing the data acquired in the calibration phase, such that the signal of interest is emphasized and any noise is attenuated. The technical aspects of preprocessing are given in detail in section 2.3. Subsequently, classifiers are trained on this data as outlined in in section 2.4.1. Training is expected to take about 2 minutes.

2.2.4 Testing Phase

The testing phase consists of applying the trained classifiers on data acquired on-line while the subject plays the Cybathlon BrainRunners game (Cybathlon, 2016). In BrainRunners, a BCI player's avatar runs along a 1D parcours composed of different platform types. It is the user's task to reach the goal platform as quick as possible, which can be achieved by actively generating the correct movement signal on the corresponding platforms. However, BrainRunners' in-game mechanics also punishes incorrect commands, by slowing down the progress of the player's avatar.

At the start of the phase, each participant received the following standardized instructions:

"Dear ⟨ Name ⟩, well done thus far! Next you will play the game BrainRunners, where your avatar moves along a parcours and the task is to reach the goal quickly. Your avatar moves quicker towards the goal by performing the correct movement on coloured platforms: Jump (violet), Roll (yellow), Speed (blue), Walk (gray). Please focus on the center of the screen and vividly imagine movement of either your left hand, right hand or feet, respectively.

Please notice how the subject is no longer cued to think of any explicit kind of feedback signal. It is assumed that the BCI misclassifying user intentions, and thus the avatar performing the wrong movement, is cause enough to generate an ErrS (Pierre W. Ferrez, 2008). At the end of the phase, each participant is asked to report on his or her experiences while testing, as outlined in section 3.4 together with all other experimental results of this block. On average, on run in the testing phase lasts about 3 minutes.

2.3 Signal Preprocessing

Our BCI system makes use of two different types of neuronal signatures described in section 2.3.1, which require the corresponding preprocessing outlined in section 2.3.2, in order to be ready for classification as described in section 2.4. The signal types of interest are as follows:

2.3.1 Neuronal Signals of Interest

Imagined Movement (Hill et al., 2006)
Event-related (De-)Synchronization (ERDS) is an induced response that indicates the subject's intention to move a body part. Changes to the signal frequency are in the bandwith of μ or β, i.e. within a window of about 8-22 Hz. lateralized central position, close to sensory-motor cortex.

Error-related Signal (Chavarriaga et al., 2014)
The class of Error-related signals (ErrS) (Chavarriaga et al., 2014) provides a neuronal signature of error, and a potential proxy of misinterpreted BCI commands. Here we outline the two most relevant error-potentials: Error-related Negativity (ERN) and Feedback-related Negativity (FRN), and argue in section 3.1 that the exact type of recorded error-signal should actually be an FRN:

Error-related Negativity (ERN) is an evoked response that indicates the subject processing an erroneous decision or choice (Ventouras, Asvestas, Karanasiou, & Matsopoulos, 2011). It occurs time-locked to an action event, with a 100 msec delay and a fronto-central scalph topography.

Feedback-related Negativity (FRN) is an evoked response that indicates the subject processing negative feedback to her or his former actions (Ward et al., 2013). It occurs time-locked to a feedback stimulus, with a 250 msec lag and a fronto-central to parieto-central scalp topography.

2.3.2 Preprocessing

To emphasize the signals of interest outlined in section 2.3.1 and to attenuate unassociated noise in the recorded EEG measurements, the raw data needs to be preprocessed. The following paragraphs provide an outline of our employed preprocessing pipeline:

Detrending Over time, channels may exhibit a baseline drift of their recorded signals, due to e.g. changes in electrode conductance or other signal non-stationarities. To cope with this issue, every electrode channel needs to be detrended—that is, a linear fit is regressed through each channel's activity time series. This linear fit is subsequently subtracted from each channel's raw data.

Bad Channel removal To identify noisy or poorly conducting channels, any channel with a power ± 3 standard deviations or more away from the mean power over all channel data is considered to be an outlier and subsequently removed from the set of considered channels.

Re-referencing Due to volume conduction or external noise interference, channels might share common information distributed over space. Common average referencing removes any joint information (of potentially external, non-neuronal origin) by subtracting the mean signal over all channels from each channel.

2.3.3 Feature processing

Feature processing is an important part of the preprocessing pipeline and pertains to extracting, selecting and normalizing the characteristics of the signals of interest. It is composed of the following steps:

Feature extraction This step applies to the imagined movement, as ERDS is an induced signal not phase-locked to any event and thus cannot be analyzed in the time domain. Therefore, we make use of Welch's method (Welch, 1967) to project the data into the frequency domain.

Feature selection This step is to constrain the information forwarded to the classifiers to exclusively match our signals of interest. As only low-frequency signals are of interest in this study, a low-pass spectral filter is applied to the data. The data for the FRN classifier was band-pass filtered between 1 - 40 Hz, and the data for the IM classifier was band-pass filtered between 8-20 Hz.

Feature normalization Feature normalization removes variability in the measurements under the assumption that such changes affect the absolute feature values, but not the relative feature values between classes. We perform an offset+scale adaptation of the feature characteristics by z-standardizing the data via its distribution parameters of mean and standard deviation.

2.4 Signal Classification

2.4.1 Offline Classifier Training

Offline training of the classifier is done on the data after preprocessing it as reported in section 2.3.2. For each of the three imagined movement conditions 20 trials are collected in the calibration phase. Furthermore, there is an additional 60 resting conditions, one per trial before the onset of the movement condition. These different numbers of class occurrence in the training data are accounted for in the classifier training procedure by balancing the class weights via scikitlearn's built-in $class\_weight =' balanced'$ option, that is similarly implemented in the widely distributed LIBSVM library(Chang & Lin, 2001) . The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * bincount(y)).

Support vector machine The support vector machine (SVM) with hinge loss is a standard choice for a classifier in the BCI literature (Dornhege, 2007). We decided to use this classifier for its quick training & cross-validation run times and the solid performances on the validation set as outlined in section 3.2. The support vector machine received as input a 2D matrix containing $[epochs \times features]$, where the features where $[time \times channels]$ concatenate to form a vector. It is well known that support vectors are sensitive to the ratio of samples to features. Informal testing of the cross-validation performance showed that this would not be an issue (see section 3.2 due to the quality of the extracted signals.

Cross-validation In order for the classifier to generalize well from the known training data to novel testing data, we performed a 10-fold cross-validation and grid search over the SVM's kernel and hyper-parameter C (which controls the margin of fit between the features and the support vectors). The cross-validation results are as reported in section 3.2. The SVM with best validation performances are subsequently used for online classification as outlined below.

2.4.2 Online Classifier Testing

For *asynchronous* imagined movement classification, the BrainRunners BCI has to decode a total of *four* different class values, occurring at an unknown point in time. In comparison, an Errs potential is *binary* (that is, it is either present or not) and occurs within a *fixed* time interval, briefly after a possibly incorrect command has been generated in-game. This makes ErrS detection a fundamentally easier problem to solve than imagined movement decoding: For four potential outcome types, the theoretical chance level classification performance is at 25%, while it doubles to 50 % for only two class values. Previous research has demonstrated that ErrS are detectable on a single-trial basis well above chance level (Llera, Gómez, & Kappen, 2012), setting the expectation that ErrS can be useful in correcting a primary movement classifier.

Imagined movement signal classifier Due to the BrainRunners player autonomously sending a self-initiated motor command at any non-fixed point in time, online classification of imagined movements is asynchronous. Self-initiated,time windows of 600msec width, describe the weighting of time points, independent BCI

Error-related signal classifier The error-related signal is an evoked potential in response to negative feedback as described in section 2.3.1. However, during the in-game testing phase, it is not known whether and when negative feedback is provided. As a proxy to this information, the brief 800 msec time window after the buffer sent a movement command to the game is classified for the presence or absence of an error-related signal.

If no error-related signal signal is detected, then the decoded imagined movement is assumed to be correct and no further action is performed. If an error-related signal signal is detected, then this indicates the previously sent movement command to be wrong and that it needs to be corrected. Automatic correction of an erroneous command may be achieved by sending the correct command on the corresponding platform. An educated guess for the correct command would be the imagined movement classifier's second most likely movement prediction. The BCI research literature promises that ErrS can be decoded on a single-trial basis well above chance level (Chavarriaga et al., 2014), and our cross-validation results reported in section 3.2 confirm a solid classification performance. Importantly, we show that the ErrS classifier's false alarm rate is expected to be very low: This is a critical result, as false alarms might undo the imagined movement classifier's correct prediction. The results in section 3.2 provide evidence that this worst-case scenario is very unlikely to happen throughout subjects, and thus the combination of an imagined movement plus an ErrS classifier should only improve on the primary classifier's performance.

3 Results

This section provides evidence that our experimental design as well as the preprocessing and classification pipeline work as intended. Furthermore, we demonstrate a working BCI that provides a brain-signal controlled interface to the Cybathlon BrainRunners game.

3.1 Validation of measured ErrS signal

An important aspect of a successful BCI is to have an experimental design which generates a signal salient enough to be detectable and decodeable. Fig. 3 illustrates the trial-averaged ErrS signal and it exhibits the properties of a stereotypical FRN response as reported in the literature: it is most salient fronto-central to parieto-central (about the position of electrode Pz), with a pronounced negative deflection about 250 msec after the feedback stimulus (Hauser et al., 2014).

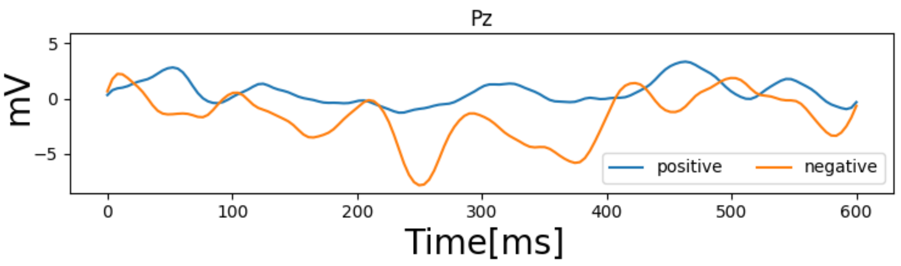


Figure 3: Trial-averaged FRN signal. The evoked potential is most present

Importantly, alternative explanations of the recorded signal being of different origin can be ruled out: Other likely signal candidates are the P300, the N300, the Mismatch Negativity and the ERN. P300, N300 and MMN can be ruled out on basis of occurring primarily occipital, and the first candidate even having the wrong polarity. Moreover, P300 and MMN are functionally related to responding to an infrequent stimulus. Though the negative-to-positive ratio of provided feedback in our calibration phase makes the negative feedback the less frequent stimulus, our chosen 1:2 ratio hardly makes it a rare oddball. Finally, though the ERN is closely related to the FRN, the experimentally measured signal occurred simply too late to be considered an ERN, which shows up at about 100 msec and evokes a primarily frontal negativity. Therefore, we argue that all but one candidate event-related potential are unlikely to be the origin of the measured signal and conclude that our calibration design does indeed evoke FRN.

Table 1: Comparison of most likely candidates of the ERP measured

	P300	N300	MMN	ERN	FRN
function	✗	✗	✗	✓	✓
polarity	✗	✓	✓	✓	✓
latency	✓	✓	✓	✗	✓
topography	✗	✗	✗	✗	✓

3.2 Classifier performance on validation data

To get an estimate of how well the classifiers might perform on the unseen testing data acquired during the BrainRunners game, it is instructive to consider the classifier performances on the hold-out data during cross-validation as summarized in table 2 for the imagined movement classifier, and in table 3 for ErrS.

To get further insights into the characteristics of validation performance, it is helpful to investigate the confusion matrices—here only reported for a single subject, though confusion matrices qualitatively looked very similar across participants. The exemplary confusion matrix for imagined movements is portrayed in Fig. 4. The figure shows most of the trials among the row-wise entries being located in the matrice's main diagonal, indicating mainly correct predictions.

Table 2: The validation performance for the optimal hyper-parameter and kernel function over 10 fold cross-validation on the calibration data per subject for the IM classifier.

Subject	Accuracy (%)	STD(%)	KERNEL	C
1	72	8	RBF	0.7
2	68	19	LINEAR	0.7
3	67	5	LINEAR	0.05

Table 3: The validation performance for the optimal hyper-parameter and kernel function over 10 fold cross-validation on the calibration data per subject for the FRN classifier.

Subject	Accuracy (%)	STD(%)	KERNEL	C
1	80	15	LINEAR	0.35
2	67	17	SIGMOID	5.9
3	53	15	LINEAR	3.1

Remarkably, rest conditions are classified perfectly and never get confused with any other movement condition—yet the discriminability between the remaining classes is rather poor. This might be due to: (i) the signal at rest being fundamentally different from all other kinds of movement signal or (ii) the fact that there was three times as much resting conditions than data per movement condition (i.e. 60 vs 20 epochs), thus providing more training data and allowing for better inference. To further investigate point (i), Fig. 5 provides plots of the trial-averaged imagined movement and rest signals for one subject in the frequency domain. The plots show that the resting condition is considerably different from the three other classes, having elevated mu-band power around the centro-parietal electrodes. It might be that artifacts such as e.g. eye blinks systematically occurred in the resting condition—but eye blinks should rather increase power at the frontal electrodes and the tested subjects did not report consciously experiencing any potential interference other than randomly occurring eye blinks. Thus we conclude that the reported differences in imagined movement and rest signals are in fact of neuronal origin.

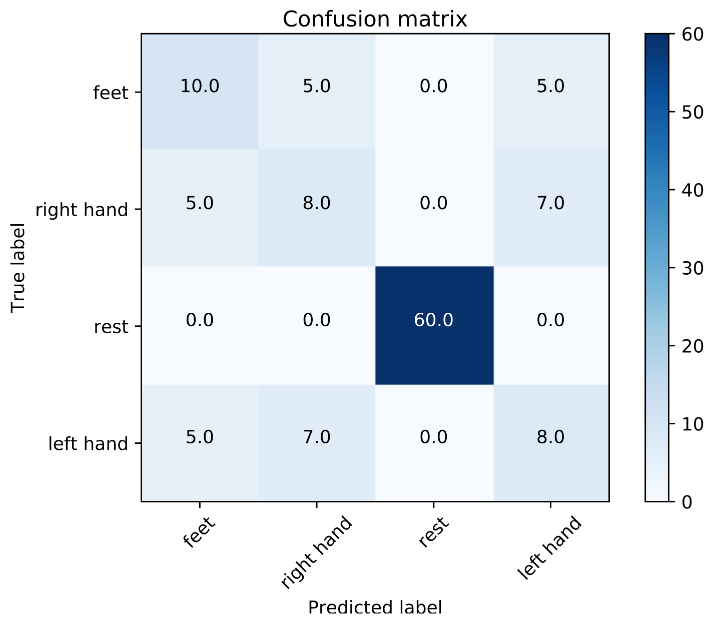


Figure 4: Confusion matrix of imagined movements for true vs predicted class labels. Entries show the absolute numbers of class occurrences and predictions. The data represented here is for subject 1.

The confusion matrix for error signals is portrayed in Fig. 6. The figure shows most of the trials among the row-wise entries being located in the matrix main diagonal, indicating mainly correct

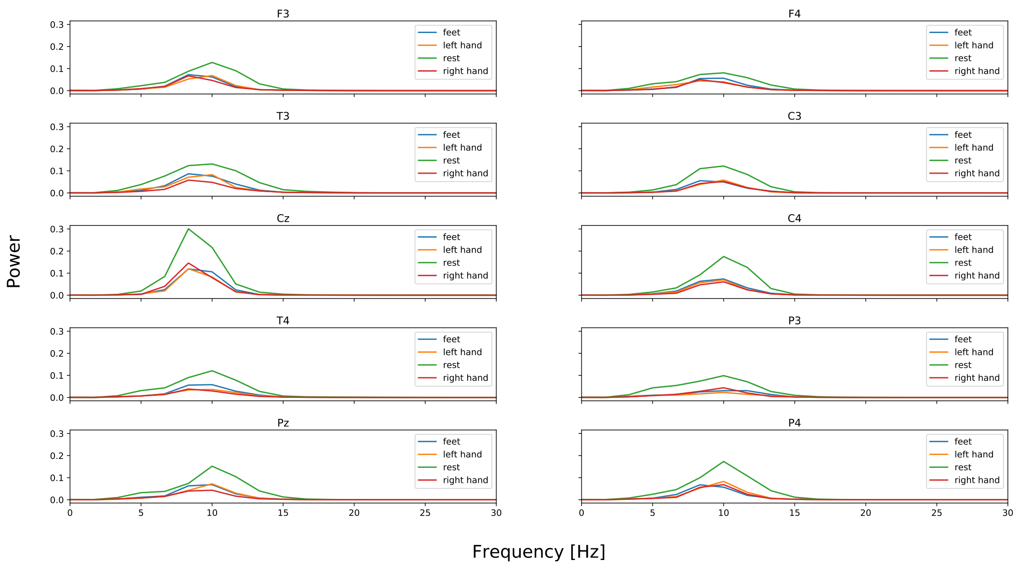
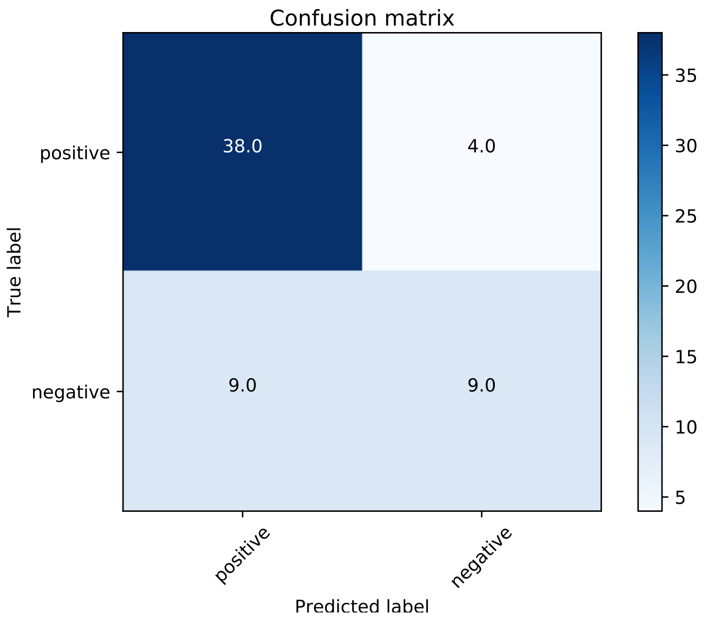


Figure 5: Power of imagined movement signals in frequency domain for the four imagined movement classes over all 10 recorded channels. The data represented here is for subject 1.

predictions. Please notice that a positive target label refers to positive feedback and the absence of an ErrS, while a negative target label implies negative feedback and the presence of an ErrS correspondingly. Therefore, the figure highlights that in case no error is committed, the secondary classifier is in fact very unlikely to report a false alarm, i.e. to say that an error occurred. This is a key finding, ensuring that the secondary classifier is unlikely to undo the correct predictions of the primary imagined movement classifier.



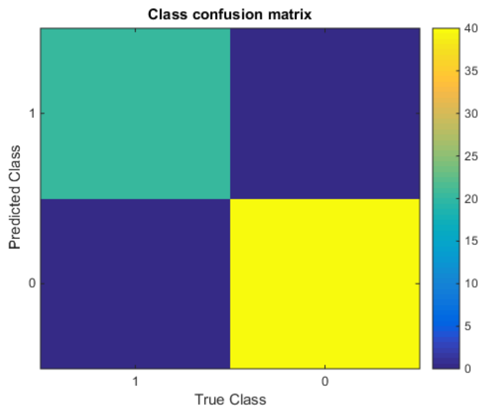
True label \ Predicted label	positive	negative
positive	38.0	4.0
negative	9.0	9.0

Figure 6: Confusion matrix of error signals for true vs predicted class labels. Entries show the absolute numbers of class occurrences and predictions. The data represented here is for subject 1.

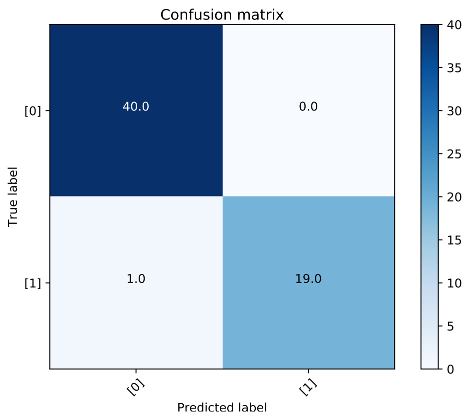
3.3 Validation of preprocessing and classification pipeline

In order to validate our preprocessing pipeline we compare it with the pipeline provided in matlab. For this, the example data set located in `matlab/offline/` was loaded and the calibration performance of the BCI scripts provided in the matlab subfolder was checked against our pipeline written in python. The parameters of the preprocessing in the matlab scripts were set such that it matched the parameters used in our BCI (section 2.2.1).

The classification of the matlab pipeline resulted in 100 percent accuracy in a one-versus-rest coding decision function scheme. The confusion matrix is depicted in figure 7a. Running the example data through our pipeline yielded a performance of 98 ± 5 percent. Note that the individual score of the 10 fold cross validation was 1.0 for 9 validation runs and 0.83 for the 10th`fold`. The confusion matrix of the two separate pipelines are identical as can be seen by comparing figures 7a and 7b. This validates that our preprocessing pipeline runs correctly.



(a) The example data for the offline processing toolbox located in `matlab/offline/`. This dataset was used to sanity check our pre-processing pipeline. As can be seen when comparing with figure 7b the results are identical.



(b) The example data for the offline processing toolbox located in `matlab/offline/`. This dataset was used to sanity check our pre-processing pipeline. As can be seen when comparing with figure 7a the results are identical.

3.4 Testing performance

Although the cross-validation results of section 3.2 are promising, the classifier did not generalize to the BrainRunners game: For two of the three tested subjects, the primary classifier constantly predicted a single output, independent of the actual imagined movement. Consequently, these subjects were excluded from the subsequent analysis, but for the remaining participant results are provided in table 4. Importantly, the results indicate an improvement of time to task completion for the hybrid BCI (with ErrS detection) over the baseline performance of the default imagined movement BCI: The subject required an average of 5.24 seconds less time to complete a run, which amounts to a 2.5 % improvement. However, with only 3 performance measurements for each the imagined movement and imagined movement plus ErrS BCI, this difference may rather be due to statistical fluctuations than reflecting a systematic difference.

As hypothesized, additional error detection improved subject performance in the BrainRunners game. However, the overall times to task completion are higher than anticipated, which is due to the overall poor performance of the imagined movement decoding. Yet, the reported subject stated that it felt in control of left-hand versus right-hand discrimination, but stated problems with the feet and rest conditions. Looking at which commands were sent to the game, this observation is underlined—the roll command as well as the speed command can be observed when the player is positioned on the respective platform, while the commands sent on the rest and jump platforms are varying and not corresponding to the platform type.

	Average runtime over 3 runs	Standard deviation
No Input	181.49	0
Bot Random	186.18	12.03
Human IM	206.47	11.07
Human IM +	201.22	2.42

Table 4: BrainRunner times-to-task-completion per simulation/experimental condition. The *No Input* condition is the baseline performance, i.e. not providing the game with any input. *Bot Random* indicates the results acquired for generating random commands every 3 seconds. Human IM and Human IM+ indicate the data collected from real test subjects.

The poor imagined movement classification performance may e.g. be due to the recruited participants being incompatible with BCI applications in general, or the employed experimental design being suboptimal. As we did confirm to find a identifiable error-related signal in all of our subject (e.g. see figure 6), we concluded that the fake-feedback used in the BCI setup is adequate. Additionally, the imagined movement task we used here is identical to the standard setup of the matlab scripts. Since other research groups have successfully applied the imagined movement task to obtain classifiable signals, we believe that the brain signals obtained from our participants were not strong enough to be decoded with success in the BrainRunner game. It is important to note that in all of our subjects the FRN classifier performed on par or better than the imagined movement classifier on the validation data, indicating a proof of concept for the initial idea to boost performance of a BCI with the aide of ErrS.

With regards to the final stage of our pipeline, we also ruled out the possibility that a different classifier may yield a significant boost in performance, since the spectral plots and time-frequency decompositions showed high similarities between the imagined movement conditions. Additionally, Support Vector Machines are a state of the art methodology which previously demonstrated good results when used in Buffer BCI's Matlab pipeline (Farquhar, 2016).

4 Discussion

In this report, we investigated the research question whether ErrS are helpful in improving the decoding accuracy and robustness of a practical EEG-based Brain-Computer-Interfacing system. As a case study, we attempted to decode imagined movements in the Cybathlon BrainRunners game.

We provided evidence that by applying a BCI composed of two classifiers, one for decoding imagined movements and another for detecting ErrS, one can achieve better performance in the Cybathlon BrainRunners game—as compared to the baseline performance of only decoding imagined movements. However, the participants of our study demonstrated an overall poor control of the imagined movement BCI, making it hard to assess whether there is indeed an absolute benefit of the added ErrS decoder. Therefore, future research needs to address this issue first and recruit a sufficient number of subjects, to provide

Furthermore, it might prove instructive to consider alternatives to or variants of the error-evoking stimuli currently used in the calibration phase. As a plus, the current experiment is designed such that ErrS data can be acquired without much added time. However, alternative experiments closer in design to the testing phase could readily accustom the subject to e.g. a feedback stimulus more similar to the one in the actual BrainRunners game.

Finally, further studies should investigate whether it is helpful to use ErrS not only to correct a single error command, but to also update the imagined movement classifier. A previous study (Llera et al., 2012) has demonstrated the feasibility of this kind of on-line classifier adaptation based on error-related signals. However, this work did only consider binary valued classes. Future research might investigate whether it is beneficial to on-line re-train the imagined movement classifier with novel incoming data: Any imagined movement prediction that elicited an ErrS in-game may be interpreted as a mislabeled data point, which is to be copied three times and labeled once with each of the other classes' target values. The re-training of the SVM would subsequently shift the decision boundary in favor of the three other classes and make future misclassifications of similar data points less likely. Though this form of classifier updating is of a rather heuristic nature and not based on any analytic closed-form solutions (as previously proposed in (Llera et al., 2012)), we are confident that our ErrS detector is robust enough to false alarms to allow for some form of updating the primary imagined movement classifier.

In sum, we provided a proof of concept that incorporating additional, passively generated brain signals might be beneficial: decoding both actively imagined movements and passively generated error signals may make mental state decoding more robust.

5 Supplementary Material

5.1 User Guide

The following user guide provides an illustrative tutorial on how to start-up and run the experiment from an experimenter perspective. To enhance user experience, the program can run in **automatic** and **manual** mode. In automatic mode, all the necessary code is started from a single main script. In manual model, the modules have to be called separately. The following passages will describe both processes individually in greater detail.

5.1.1 Pre-requisites

This BCI system assumes that one has python 2.7 installed and is correctly located in the PATH variable of your OS. The system was developed on Zorin OS, a Ubuntu fork. Additionally, the BCI was tested on Windows 10, code is present to run on Mac but due to limited Mobita support this was not fully tested. For any errors, manual mode will run regardless of the OS being used.

5.1.2 Automatic Mode

The program flow is depicted in figure 8. Before starting the automatic mode, be sure that a buffer is started. If one is debugging please start the `./debug_quickstart.bat` on windows and `./debug_quickstart.sh` on Linux / Mac. These terminal scripts are located in `Buffer/`. When using the Mobita run `./eeg_quickstart.bat` , `./eeg_quickstart.sh` on Windows, or Linux / Mac respectively.

Running `Project/Code/run.py` will started the essential script for the BCI system. The user is greeted with a welcome screen, and the experimenter should initialize the calibration phase by pressing 1 (see figure 8). After the calibration phase, the classifier is automatically trained. Check the output in the terminal for the classifier performance. Additionally one can run the `visualize.py` script to view the ERPs, and time frequency plots by inputting the subject number of the user (see below). By pressing 2 on the main screen the Cybathlon phase starts, the user is greeted with a player choice, choice of version (imagined movement, imagined movement plus error detection) followed by an instruction page. The game should then be started by the experimenter.

5.1.3 Checklist automatic mode

Be sure that the following is executed in order to use the BCI system:

1. is a buffer running, i.e.. `./debug_quickstart.bat/sh` or `./eeg_quickstart.bat/sh`?
2. did you start `run.py`?

Note: this assumes python 2.7 is in your path, `run.py` uses they symbolic variable 'python' to point to python 2.7. If this is not the case either edit the corresponding lines in the script, or go for manual mode.

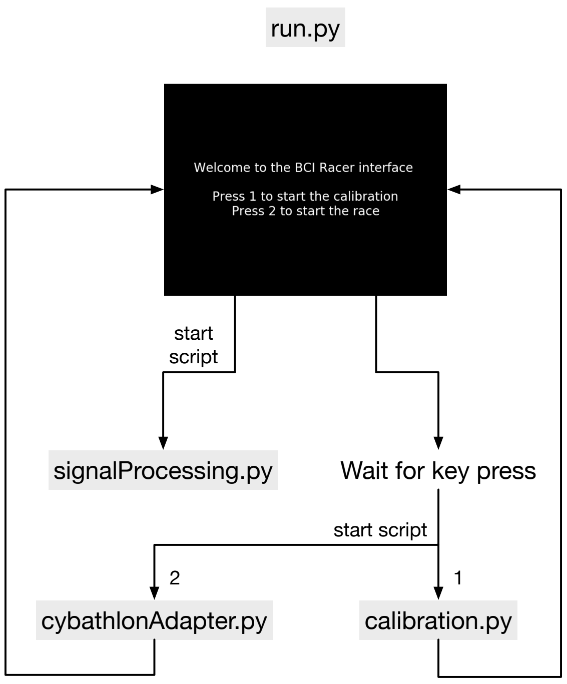


Figure 8: The flow of the program in automatic mode.

5.2 Manual Mode and Programmer's guide

The following section will describe how to start the individual parts of the BCI manually. Automatic mode is initialized with `run.py`. Manual mode can be run with either the Mobita or in debug mode. In order to start debug mode one needs to start `./debug_quickstart.bat` on Windows, and `./debug_quickstart.sh` on Linux / Mac. When connecting to the Mobita, run the `./eeg_quickstart.bat` on Windows, and `./eeg_quickstart.sh` on Linux / Mac. The following code should be run separately regardless of the mode:

1. `moduleInstaller.py` a script that will install all the modules necessary for running the other code in the BCI. Please note that this uses python 2.7, python 3 is not supported.

2. `signalProcessing.py` a script that controls the calibration, train and test phase of the experiment. The script interfaces with the FieldTrip buffer and listens to events that cause it to switch between the calibration, train, and test phase.

   (a) When `start calibration` is sent, signalProcessing.py listens to the trigger events `feedback`, and `target`. After it receives `calibration end`, it will initiate the standard pre-processing pipeline and store all the data to disk.

   (b) When `start training` is received from the buffer, signalProcessing.py will load the calibration data from disk and train the IM classifier and FRN classifier.

   (c) When `start test` is received, `signalProcessing.py` will switch to either imagined movement only mode, or imagined movement + error detection mode. The former will cause `signalProcessing.py` to only catch 3 seconds of data from the buffer, consequently producing the maximal prediction from the IM classifier. In contrast, the latter will first catch 3 seconds of data from the buffer, and then another 600 msec from the buffer for the FRN classifier to make a prediction.

   Additionally, this script saves the calibration and test data to disk.

3. `calibration.py` controls the calibration phase as outlined on section 2.2.2. At the top of this script one can edit the experimental settings, specifically one can edit the variables:

   (a) `nTrials`, sets the number of trials

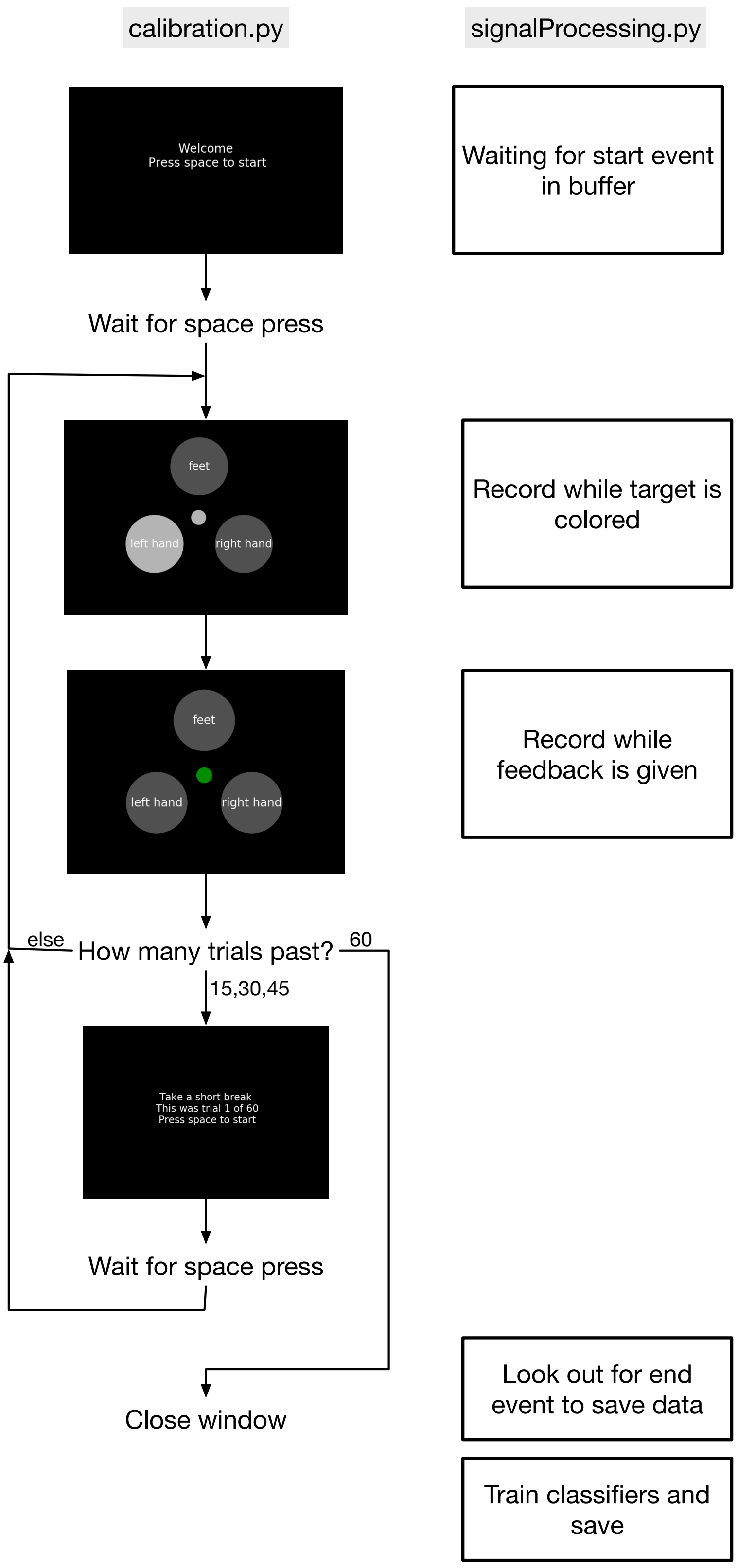
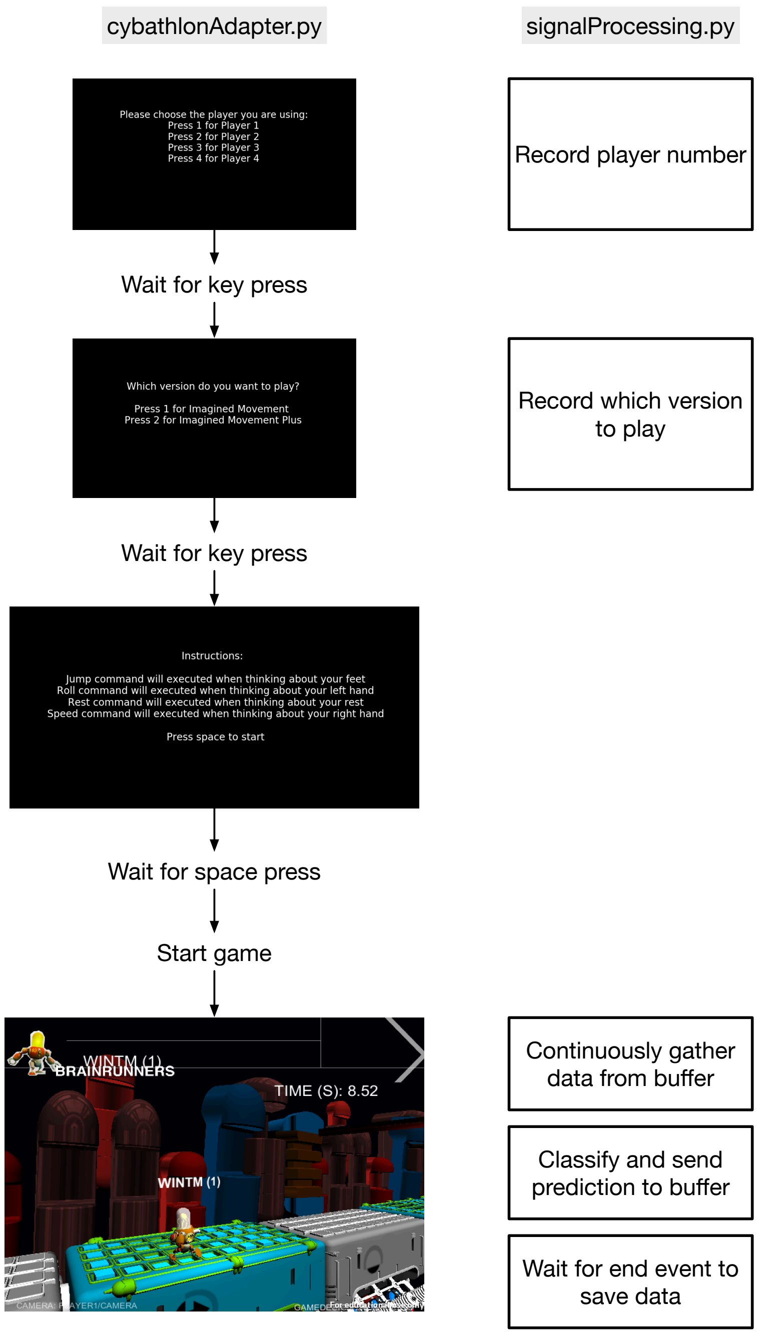


Figure 9: The program flow of the calibration phase.



```
Please choose the player you are using:
Press 1 for Player 1
Press 2 for Player 2
Press 3 for Player 3
Press 4 for Player 4
```

```
Which version do you want to play?

Press 1 for Imagined Movement
Press 2 for Imagined Movement Plus
```

```
Instructions:

Jump command will executed when thinking about your feet
Roll command will executed when thinking about your left hand
Rest command will executed when thinking about your rest
Speed command will executed when thinking about your right hand

Press space to start
```

Figure 10: The program flow of the game phase.

   (b) `dt`, sets the time step
   (c) `targetDuration`, sets the target condition duration
   (d) `restDuration`, sets the rest condition duration
   (e) `proportionNegative`, sets the proportion of negative feedback trials
   (f) `breakTrial`, sets after how many trials a break is inserted

4. `cybathalongAdapter.py` a script that interfaces with the game. Specifically, it catches predictions from the classifier and sends the corresponding command to the brain runners game. Note, this script can be edited to control such that the buffer and the game can be at different IP addresses. Specifically the variables :

```
# Configuration of buffer
buffer_hostname = 'localhost'
buffer_port     = 1972

# Configuration of BrainRacer
# br_hostname='131.174.105.190'
br_hostname     = 'localhost'
br_port         = 5555
```

   control where the buffer is located, and where the Brain Racer game sends events from.

Next to these core script there is helper code for training the classifier, data overwrite protection, preprocessing etc. All these scripts have an independent mode, i.e. they can load calibration data from `Project/Data` and implement the functions of the corresponding modules in section 2.3.2 or they are visualization functions.

1. `classification.py` contains the information of the classifier. K-fold cross-validation is used (default value is K = 10) in combination with a grid search to find the optimal parameters for the calibration data. The grid search is over the decision functions, i.e. radial basis function (rbf), sigmoid, and linear, and the C parameter which controls the margin of the decision plane. This script can be run independent of the other scripts. One can input at the bottom of the script a dataset (located in `Project/Data/`. The script will output the optimal model over this calibration data, in combination with a confusion matrix over the K validation sets. This script can be extended to contain other classifiers, to fit the experimenter's need.

2. `preproc.py` is in control of the preprocessing outlined in section 2.3.2.

3. `visualize.py` allows for plotting event related potentials and time frequency plots using complex wavelet convolution. Note, ones can edit the subject number at the bottom of the script to view the ERP and or time frequency decomposition for the subject of interest.

4. `systemHelper.py` contains the module `checkOverwrite` which will check whether a datatype is already present in a subdirectory. This module is run everytime data is stored in `signalProcessing.py`. The module can check any file type parsed to it, see function description for more information.

5. `visualize.py` contains plotting functions that can produce :
   (a) Confusion matrix
   (b) Event related potentials per class indicated in events
   (c) Time frequency plots per class per channel averaged over class
   (d) Spectral plots using Welch's method

   This script is used to produce the plots for the report.

5.2.1 Checklist for manual mode

Be sure that the following is executed in order to use the BCI system:

1. start a buffer, i.e. `./debug_quickstart.bat/sh` or `./eeg_quickstart.bat/sh`
2. start `signalProcessing.py`
3. for calibration, start `calibration.py`
4. for playing BrainRunner start `cybathalonAdapter.py`

References

Blokland, Y., Spyrou, L., Lerou, J., Mourisse, J., Jan Scheffer, G., van Geffen, G.-J., ... Bruhn, J. (2015). Detection of attempted movement from the EEG during neuromuscular block: proof of principle study in awake volunteers. *Scientific Reports*, *5*(August), 12815. Retrieved from `http://dx.doi.org/10.1038/srep12815{%}5Cnhttp://www.nature.com/doifinder/10.1038/srep12815` doi: 10.1038/srep12815

Chang, C.-c., & Lin, C.-j. (2001). LIBSVM : A Library for Support Vector Machines. *Tist*. Retrieved from `http://www.csie.ntu.edu.tw/{~}cjlin/libsvm/` doi: 10.1145/1961189.1961199

Chavarriaga, R., Sobolewski, A., & Millan, J. d. R. (2014). Errare machinale est: The use of error-related potentials in brain-machine interfaces. *Frontiers in Neuroscience*(8 JUL). doi: 10.3389/fnins.2014.00208

Cybathlon. (2016). *Cybathlon Brain-Computer Interface Race : Manual for Cybathlon Brainrunner-sTraining.* Zurich: Zurich University of Arts, Specialization in Game Design.

Dornhege, G. (2007). *Toward brain-computer interfacing.* MIT Press.

Farquhar, J. (2015). *Mobita Manual.* Retrieved from `https://github.com/jadref/buffer{_}bci/blob/master/doc/Mobita{_}manual.md`

Farquhar, J. (2016). *Buffer BCI.* Retrieved from `https://github.com/jadref/buffer{_}bci`

Hauser, T. U., Iannaccone, R., Stümpfli, P., Drechsler, R., Brandeis, D., Walitza, S., & Brem, S. (2014). The feedback-related negativity (FRN) revisited: New insights into the localization, meaning and network organization. *NeuroImage*, *84*, 159–168. doi: 10.1016/j.neuroimage.2013.08.028

Hill, J., Lal, T. N., Schroeder, M., Hinterberger, T., Widman, G., Elger, C., ... Birbaumer, N. (2006). Classifying Event-Related Desynchronization in EEG, ECoG and MEG signals. *Pattern Recognition*, *4174*, 404–413. Retrieved from `http://link.springer.com/chapter/10.1007/11861898{_}41{%}5Cnhttp://eprints.pascal-network.org/archive/00002127/` doi: 10.1007/11861898_41

Llera, A., Gómez, V., & Kappen, H. J. (2012). Adaptive classification on brain-computer interfaces using reinforcement signals. *Neural computation*, *24*(11), 2900–23. Retrieved from `http://www.ncbi.nlm.nih.gov/pubmed/22845827` doi: 10.1162/NECO_a_00348

Oostenveld, R., Fries, P., Maris, E., & Schoffelen, J. M. (2011). FieldTrip: Open source software for advanced analysis of MEG, EEG, and invasive electrophysiological data. *Computational Intelligence and Neuroscience*, *2011*. doi: 10.1155/2011/156869

Parra, L. C., Spence, C. D., Gerson, A. D., & Sajda, P. (2003). Response error correction-a demonstration of improved human-machine performance using real-time EEG monitoring. IEEE Transactions on. *Neural Systems and Rehabilitation Engineering*, *11*(2), 173–177.

Pierre W. Ferrez, J. D. R. M. (2008). Error Related EEG Potentials Generated During Simulated Brain-Computer Interaction. *IEEE Transactions on Biomedical Engineerin*, *55*(3), 923–929.

Van Gerven, M., Farquhar, J., Schaefer, R., Vlek, R., Geuze, J., Nijholt, A., ... others (2009). The brain–computer interface cycle. *Journal of neural engineering*, *6*(4), 041001.

Ventouras, E. M., Asvestas, P., Karanasiou, I., & Matsopoulos, G. K. (2011). Classification of Error-Related Negativity (ERN) and Positivity (Pe) potentials using kNN and Support Vector Machines. *Computers in Biology and Medicine*, *41*(2), 98–109. doi: 10.1016/j.compbiomed.2010.12.004

Ward, T., Bernier, R., Mukerji, C., Perszyk, D., McPartland, J. C., Johnson, E., ... Perszyk, D. (2013). Feedback-Related Negativity. *Encyclopedia of Autism Spectrum Disorders*, 1256–1257. Retrieved from `http://www.springerlink.com/index/10.1007/978-1-4419-1698-3{_}731` doi: 10.1007/978-1-4419-1698-3_731

Welch, P. D. (1967). The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms. *IEEE Transactions on audio and electroacoustics*, *15*(2), 70–73. doi: 10.1109/TAU.1967.1161901