h e p i a

Haute école du paysage, d'ingénierie
et d'architecture de Genève

Brain Interface Terminal

hepia - 2011

LESCOURT Adrien

Table des matières

1 EmotivLib

1.1 Description

La librairie `emotivLib` a été initialement développée par Jonathan Froidevaux dans le cadre d'un travail de Bachelor à la HE-Arc. Nous avons pris cette librairie comme point de départ à nos travaux sur le casque `EPOC`. Au fur et à mesure de nos travaux, nous avons du modifier et adapter cette librairie pour qu'elle satisfasse nos besoins. Nous expliquerons donc dans ce document quelles sont ces modifications et comment utiliser efficacement la dernière version de cette librairie.

1.1.1 Utilisation

L'utilisation de la librairie reste identique : Il faut instancier la classe `EmotivController` et les données provenant du casque sont toujours stockées dans les dictionnaires définis par l'interface `IDataList`. Cependant cette interface est désormais composée de trois dictionnaires, accessible par les méthodes suivantes :

- `GetModalities()`
- `GetInfos()`
- `GetRawData()`

Le premier tableau contient toutes les informations pré-traitée du casque (émotif, cognitif, gyroscopique et musculaire). GetInfos() ne retourne plus que des informations directement liée à l'utilisation du casque (batterie, état des contacts...). Et `GetRawData()` les données bruts du casque.

Toutes les clés de ces dictionnaires sont disponible à la table 1.1

1.1.2 Données bruts

Nos recherches nous ont orienté vers la recherche d'Event Related Potential (ERP), en particulier le P300. Il était donc nécessaire de récupérer les données bruts de chaque capteur du `EPOC`. Nous nous sommes alors aperçu que ces informations (qui étaient alors stockées dans le dictionnaire `getInfos()`) étaient enregistrées à une fréquence extrêmement faible (de l'ordre de 10Hz).

Le fonctionnement du casque est le suivant : pour transmettre les informations du casque vers l'utilisateur, il faut appeler la méthode `ProcessEvent` de l'instance du casque (`emoEngine`). Même en réduisant le temps entre deux appels successifs de cette méthode, la fréquence ne dépassait pas 10Hz. Il s'avère en réalité que le casque stock dans un buffer toutes les données capturées (la taille maximum de ce buffer est de 128 éléments). La méthode d'acquisition telle qu'elle était implémenté ne tenait pas compte de ce buffer et n'extrayait que la première ligne à chaque acquisition.

Le dictionnaire obtenu par `GetRawData()` comporte désormais pour chaque clé un tableau de `double` comportant toutes les valeurs capturées par le casque. On arrive ainsi à la fréquence d'acquisition du EPOC : 128Hz.

Par ailleurs, un évènement `EmotivController.OnDataChange` est déclenché à chaque nouvelle entrée dans ce dictionnaire.

1.1.3 Exemple d'acquisition

Le code 1.1 représente l'utilisation type de `emotivLib` pour l'acquisition des données.

Listing 1.1– Exemple type d'aquisiation via emotivLib

```
using EmotivLib;
using DataList;

public class EmotivControl : EmotivController
{
    public Dictionary<string, double> ModaVal = new Dictionary<string, double>();
    public Dictionary<string, double> InfoVal = new Dictionary<string, double>();
    public Dictionary<String, double[]> RawData = new Dictionary<String, double[]>();

    public EmotivControl() : base()
    {
        OnDataChange += EmoControllerOnDataChange;
        Start();
    }

    private void EmoControllerOnDataChange(object sender, IDataList data)
    {
        ModaVal = data.GetModalities();
        InfoVal = data.GetInfos();
        RawData = data.GetRawData();
    }
}
```

Il faudra bien évidemment protéger les accès concurrents pour ces trois dictionnaires.

1.1.4 Données cognitives

Le casque `EPOC` permet de détecter jusqu'à quatre motifs cognitifs. Emotiv possède la liste des actions cognitives utilisées dans un registre accessible en lecture/écriture par respectivement
`CognitivGetActiveAction()` et `CognitivSetActiveAction()`.
Pour permettre l'utilisation des motifs cognitifs, il est impératif de mettre le bit correspondant à '1'. Emotiv a défini les actions cognitives selon le listing 1.3 (et pour retrouver les bits correspondants aux actions entraînées, on utilisera la méthode
`CognitivGetTrainedSignatureActions()`)

Le `EPOC` ne peut apprendre que 4 motifs cognitifs simultanément (+ le neutral). De plus, il est tout à fait possible d'enregistrer un autre motif que "Pousser" dans `COG_PUSH`. Nous utiliserons donc que 4

de ces actions cognitives, et avec la correspondance 1.2.

Listing 1.2– Correspondance des motifs cognitifs

```
public enum CognitivAction
{
    Cognitiv1 = EdkDll.EE_CognitivAction_t.COG_PUSH,
    Cognitiv2 = EdkDll.EE_CognitivAction_t.COG_PULL,
    Cognitiv3 = EdkDll.EE_CognitivAction_t.COG_LIFT,
    Cognitiv4 = EdkDll.EE_CognitivAction_t.COG_DROP
}
```

1.1.5 Utilisation des motifs cognitifs

Avant toute utilisation de cette détection cognitive, il est nécessaire de "nettoyer" le registre `activeAction`. Cette action doit être effectuée une fois que le `EPOC` est prêt (L'évènement `OnHeadSetReady a été déclenché`).

```
monEmotivController.EmoEngine.
    CognitivSetActiveActions(monEmotivController.UserId, 0);
```

Ensuite l'utilisateur doit entrainer l'état `neutral` pendant au moins 30 secondes. On lance ce training via `StartCogSamplingNeutral()` et on l'arrête avec `StopCogSamplingNeutral()`.

Chaque motif cognitif voulant être utilisé nécessite alors 8 secondes d'entrainement.

```
monEmotivController.StartTrainingCognitivAction(
    EmotivLib.CognitivAction.Cognitiv1);
```

A la fin de l'entrainement, l'évènement `OnCognitivTrainingCompleted` est déclenché.

Une fois toutes ces étapes établies, les valeurs de détections de ces motifs peuvent être lues dans le champs correspondant du dictionnaire retourné par `GetModalities()`.

```
double val = monEmotivController.ModaVal["Cognitiv1"];
```

1.1.6 Gestion de profil

Les différentes phrases d'entrainement étant relativement longues, il est possible de les sauvegarder dans des profils (fichier .emu). En outre, ces fichiers sont partageables entre les applications utilisant la librairie et les applications fournies par Emotiv, telle que EPOC Control Panel.

1.1.7 État des capteurs

L'état de la liaison des différents capteurs sont stockés dans le dictionnaire accessible via `getInfos()`. Ce dernier stockant des objets de type `double`, nous avons établi la correspondance suivante :

- 0.0 = aucun signal (noir)
- 0.2 = très mauvais signal (rouge)
- 0.4 = signal faible (orange)
- 0.6 = signal moyen (jaune)
- 0.8 = bon signal (vert)

TABLE 1.1 – Clé des dictionnaires de IDataList

Modalités	Informations	Données Bruts
Engagement	SignalQuality	COUNTER
Frustration	BatteryLevel	INTERPOLATED
Meditation	ContactQuality0	RAW_CQ
Excitement	ContactQuality1	AF3
LongTermExcitement	ContactQuality2	F7
DeltaGyroLeft	ContactQuality3	F3
DeltaGyroRight	ContactQuality4	FC5
DeltaGyroUp	ContactQuality5	T7
DeltaGyroDown	ContactQuality6	P7
LookRight	ContactQuality7	O1
LookLeft	ContactQuality8	O2
BlinkRight	ContactQuality9	P8
BlinkLeft	ContactQuality10	T8
Blink	ContactQuality11	FC6
EXP_NEUTRAL	ContactQuality12	F4
EXP_EYEBROW	ContactQuality13	F8
EXP_FURROW	ContactQuality14	AF4
EXP_SMILE	ContactQuality15	GYROX
EXP_CLENCH		GYROY
EXP_LAUGH		TIMESTAMP
EXP_SMIRK_LEFT		ES_TIMESTAMP
EXP_SMIRK_RIGHT		FUNC_ID
Cognitiv1		FUNC_VALUE
Cognitiv2		MARKER
Cognitiv3		SYNC_SIGNAL
Cognitiv3		BUFFER_SIZE

1.2 Annexes

1.2.1 Clés des dictionnaires de IDataList

1.2.2 Motifs Cognitifs

Listing 1.3– Motifs cognitifs définit par Emotiv

```
COG_NEUTRAL                    = 0x0001,
COG_PUSH                       = 0x0002,
COG_PULL                       = 0x0004,
COG_LIFT                       = 0x0008,
COG_DROP                       = 0x0010,
COG_LEFT                       = 0x0020,
COG_RIGHT                      = 0x0040,
COG_ROTATE_LEFT                = 0x0080,
COG_ROTATE_RIGHT               = 0x0100,
COG_ROTATE_CLOCKWISE           = 0x0200,
COG_ROTATE_COUNTER_CLOCKWISE   = 0x0400,
COG_ROTATE_FORWARDS            = 0x0800,
COG_ROTATE_REVERSE             = 0x1000,
COG_DISAPPEAR                  = 0x2000
```

2 CogniEmo

2.1 Présentation

CogniEmo est une application `WPF` développé en `C#` similaire à EmoRate [1]. L'objectif est d'utiliser les mécanismes de détections cognitives disponible via le casque `EPOC` d'Emotiv pour détecter les émotions primaires suivantes :

- Joie
- Tristesse
- Haine
- Peur

L'application utilise la librairie `EmotivLib` pour communiquer avec le casque.

2.2 Fonctionnalité

CogniEmo implémente les fonctionnalités suivantes :

2.2.1 Interface utilisateur

- L'entrainement des 4 émotions primaires et de l'état neutral.
- Une progress bar affichant la valeur de chaque émotion entraînée (Les émotions non entraînées ont leur nom barré)
- L'état des capteurs du `EPOC`
- Une gestion des profils utilisateurs. Cela permet notamment de sauvegarder/charger l'entrainement des émotions.
  - EPOC Control Panel enregistre ses profils utilisateurs dans le dossier `CommonApplicationData\Emotiv` (Sous Windows 7, cela correspond à `C:\ProgramData\Emotiv`). Pour permettre le partage de la gestion des profils avec ce logiciel, nous enregistrons aussi les utilisateurs dans ce même dossier.
  - CogniEmo force l'utilisation ou la création d'un utilisateur. Si toutefois aucun utilisateur n'est créé/chargé, l'utilisateur Default est utilisé, avec la contrainte que ce profil n'est jamais sauvegardé.

2.2.2 Contraintes d'utilisation

- Un mécanisme de vérification visant à contraindre l'utilisateur à suivre une démarche formelle
  - Impossibilité de lancer un nouvel entrainement tant qu'un entrainement est en cour (ce qui entraînerait un crash de la SDK d'Emotiv)

1. http ://www.androidreview.com/html/emorate.php

- Impossibilité d'entraîner une émotion tant que le neutral n'est pas entraîné
- Impossibilité de sauver les données ou le profil tant qu'aucune émotion n'est entraînée ou qu'aucun profil n'est chargé

2.2.3 Sauvegarde des données

L'application sauvegarde toutes les données détectées par le casque dans un fichier .csv.

2.3 Remarques

- La détection cognitive est extrêmement sensible aux parasites : Il est obligatoire de rester le plus immobile possible quand on utilise cette application.
- La détection d'émotion fonctionne beaucoup mieux avec de vraies émotions. L'activité cérébrale et musculaire n'est pas la même quand on se force à rire et quand on rit réellement.
- La phase d'apprentissage des émotions est donc la plus critique. C'est la raison pour laquelle il est indispensable d'avoir un protocole pour l'entrainement et de sauvegarder le profil une fois les résultats satisfaisants obtenus. Cependant lorsque l'on recharge un profil ultérieurement, la détection peut être moins efficace que lors de la phase d'entrainement. Cela est due au positionnement du casque : Les capteurs doivent être placées exactement aux mêmes endroits que lors de l'apprentissage.
- Cette application n'effectue pas de traitement sur les données enregistrées.

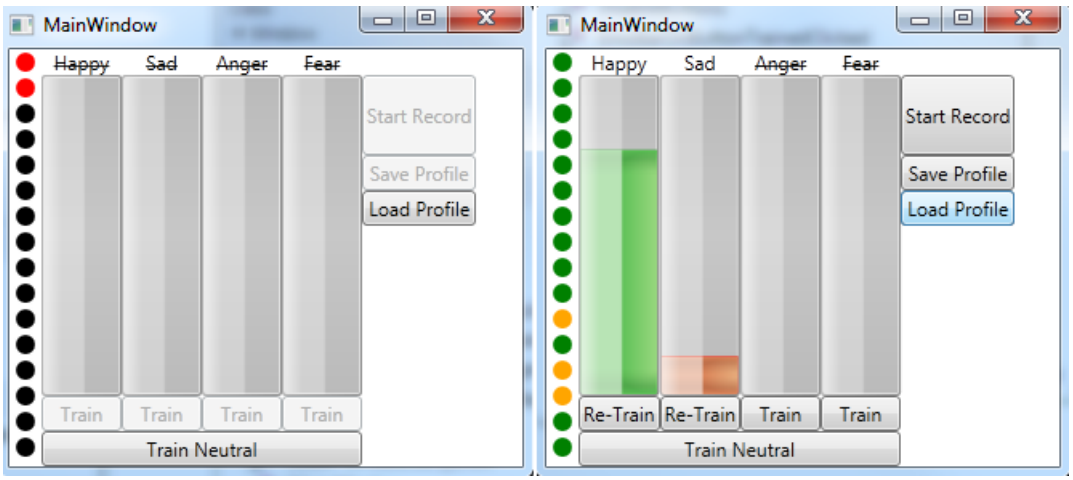

FIGURE 2.1 – CogniEmo

3 BlinkWriter

L'application BlinkWriter est une application `WPF` développée en `C#`. Son objectif est de permettre la saisie de texte par le clignement des yeux. Elle se base sur le fonctionnement de BCI Speller, afin d'être exploitable par un ERP (P300) à l'avenir. Le but sera alors de remplacer le clignement des yeux par la détection du P300.

Le Speller est composé d'un carré de 6 par 6, contenant toutes les lettres de l'alphabet ainsi que 9 chiffres et le caractère espace.

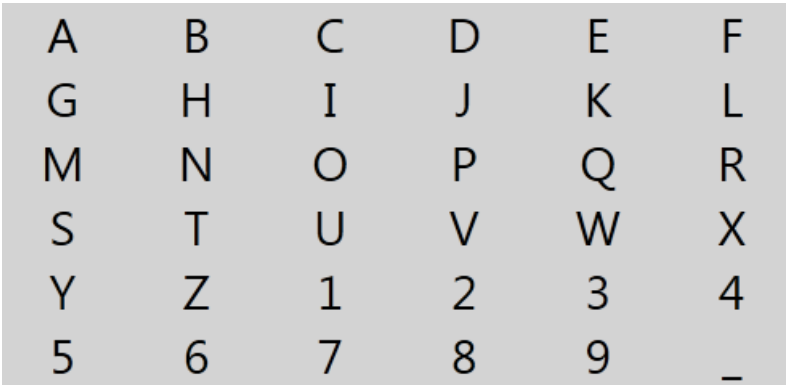

FIGURE 3.1 – Speller

3.1 Fonctionnement

Le Speller consiste à "flasher" (éclairer très vite) aléatoirement et périodiquement une ligne ou une colonne de lettres. L'utilisateur doit attendre que la lettre qu'il veut saisir soit éclairée par un de ces flashs. Dès lors, il devra immédiatement cligner des yeux. Il devra le faire deux fois : une fois quand la lettre est éclairée horizontalement et une deuxième fois verticalement (l'ordre n'a pas d'importance). En croisant la ligne et la colonne, on peut alors retrouver la lettre désirée, l'écrire et passer à la suivante.

Dans une première version, l'écart entre deux flashs est suffisamment important pour laisser le temps à toute la chaine de réaction :

- Flash de la ligne voulu
- Réaction du l'utilisateur
- Clignement des yeux
- Détection du clignement par le casque
- Transmission de l'information au programme
- Prise en compte dans le programme

Avec ce procédé, le temps de séparation entre deux flashs est environ 800 millisecondes.

Une deuxième version prend en compte le chevauchement des flashs : On accélère la fréquence de clignotements, et lors de la détection d'un clignement des yeux, on ne récupère plus la dernière ligne ou colonne flashé, mais la dernière-N. Ce N étant un offset défini en fonction du temps de réaction (Ex : si N=1, on récupère l'avant dernière ligne ou colonne flashé. Cela veut dire qu'il y a eu un flash entre le moment où le flash voulu s'effectue et la prise en compte dans le programme). Cette valeur N devra être préalablement paramétrée en fonction de la vitesse de clignotements choisis et le temps de réaction de l'utilisateur. Le temps entre les flashs est défini dans le champs `Flash Speed [ms]` et l'offset dans le champs `Blink Offset [0-3]`.

Avec cette méthode, on arrive à saisir du texte avec des flashs espacés de 300 millisecondes. Cependant, cette vitesse demande une grande attention de l'utilisateur. La saisie est alors plus fatigante qu'avec la première méthode.

Avant toute utilisation de cette application, il faudra s'assurer de la bonne détection du clignotement des yeux avec EPOC Control Panel.

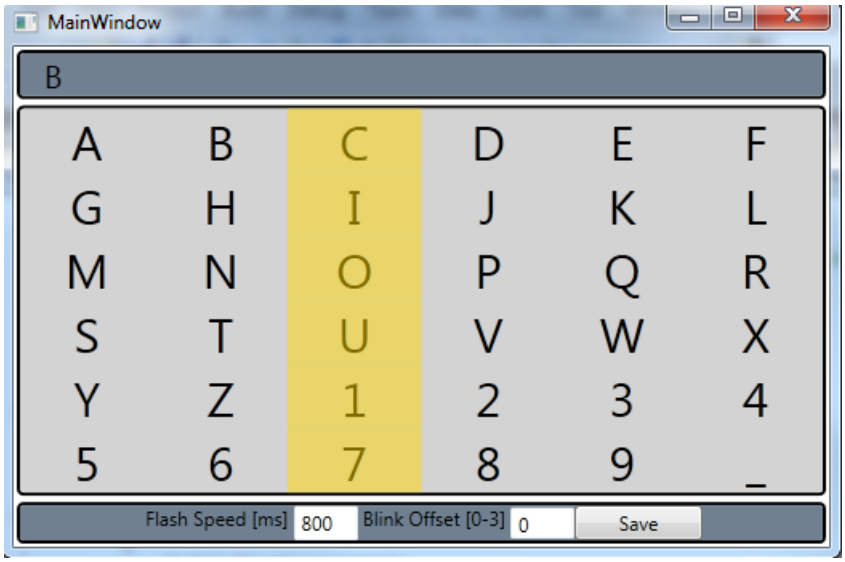

FIGURE 3.2 – Blink Writer

3.2 Documentation technique

Cette partie détaille succinctement les choix effectués pour implémenter cette application.

3.2.1 Détection du clignement

La détection d'un clignement des yeux se fait en lisant les modalités `BlinkLeft`, `blinkRight` et `Blink` (`1` s'il y a clignement, `0` sinon). Si un seul des trois est à `1`, on considère qu'il y a clignement. Les données sont lues à chaque changement de données dans le casque (sur l'event `OnDataChange`). Cela implique que pour un seul clignement, les données peuvent être lues plus d'une fois. Pour éviter de détecter plusieurs fois le même clignement, nous nous baserons alors uniquement sur les flancs montants.

3.2.2 Mécanismes de clignotement

Dans le speller (classe `WriterArea`) la représentation de la ligne ou colonne flashé est un entier : Si sa valeur est inférieure au nombre de ligne, il correspond directement à l'index de la ligne, sinon c'est sa valeur moins le nombre de ligne qui correspond à l'index de la colonne.

Pour choisir la ligne ou la colonne à éclairer, on tire aléatoirement un nombre entre zéro et la somme du nombre de ligne et de colonne moins un (pour l'instant le speller est composé de 6 lignes et 6 colonnes donc entre 0 et 11). La distribution de ce tirage pseudo aléatoire a beau être uniforme, il est intéressant dans notre cas de faire en sorte qu'au bout de 12 tirages, chaque ligne et colonne ont été éclairées une fois. Cela permet d'accélérer la vitesse générale de la saisie. Ce mécanisme a été implémenté comme si on mettait tous les chiffres de 0 à 11 dans un sac puis on les prend un par un. Lorsque que le sac est vide on le re-remplit.

A chaque flash, on stock dans un tampon circulaire son index. Le speller propose alors les méthodes permettant à chaque instant d'aller lire quel élément a été flashé en position dernier - N (N étant l'offset définit en 3.1).

```
public int CurrentFlashedRow()
public int CurrentFlashedColumn()
```

3.2.3 Détection du caractère

Une fois qu'une ligne et une colonne ont été détectées, on retrouve le caractère. On considère pour cela que la lettre est à la position POS dans la grille, telle que :

```
int POS = ligneFlash * nbrDeLigne + colFlash;
```

Les 26 premiers éléments de la grille sont les lettres :

```
string lettreSaisi = ((char)(POS + (int)'A')).ToString();
```

Les 9 qui suivent sont les chiffres :

```
string lettreSaisi = (POS - 25).ToString();
```

Le dernier élément de la grille est l'espace.

4 SearchP300

4.1 Présentation

L'application SearchP300 a pour objectif de logger certaines données bruts du casque afin de les traiter ultérieurement dans l'optique de détecter un Event Related Potential (ERP), le P300.

Le P300 se caractérise en électroencéphalographie par une variation positive de tension, approximativement 300 millisecondes après qu'un évènement attendu se produise (visuel, auditif ou tactile). Cependant, ce signal possède un rapport signal-bruit extrêmement faible, ce qui rend sa détection très difficile.

Dans son positionnement standard, le casque `EPOC` ne couvre pas la zone où le P300 sera visible. Il faut le placer légèrement en arrière afin que les capteurs `AF3`, `F3`, `AF4` et `F4` se rapproche de la zone `Cz` du système `10-20` [1].

4.2 Acquisition

La première étape réalisé pour détecter le P300 est l'acquisition des données. Le procédé mis en place est le suivant : Le programme va flasher six chiffres affichés un par un. L'utilisateur devra quant à lui focaliser son attention uniquement sur un chiffre encerclé en rouge, et ne pas regarder les différents flashs. Il doit attendre le moment ou son chiffre encerclé sera flashé (voir figure 4.1). Lorsque cela se produit, c'est un nouveau nombre qui est encerclé. Ce procédé permet d'enregistrer une suite d'évènements attendus, et donc une suite de P300. Pour permettre à l'utilisateur d'effectuer plusieurs mesures, l'ordre de flash des nombres n'est pas aléatoire, mais toujours le même d'une exécution sur l'autre. Cela permettra de procéder à des moyennes sur les mesures afin d'augmenter le ratio signal/bruit.

4.2.1 Acquisition des données bruts

Nous avions abordé la possibilité d'obtenir les tensions des capteurs via le SDK d'Emotiv. Seul les capteurs permettant potentiellement de détecter le P300 sont enregistrés : `AF3`, `F3`, `AF4` et `F4`. Trois autres champs sont aussi ajoutés à ces capteurs : `COUNTER` et `BUFFER_SIZE` servent à vérifier si les données enregistrées sont cohérentes (pas de dépassement du compteur, et buffer inférieur à 128). Enfin le `TIMESTAMP` informe à quel instant chaque mesure a été effectuée. Le format utilisé est `.csv`, le point virgule étant le délimiteur.

Puisque les différentes captures vont potentiellement être traitées en même temps, il est nécessaire de les synchroniser. Pour pouvoir faire des moyennes à partir de plusieurs prises, chaque enregistrement

1. Le système 10-20 est une méthode internationale reconnue de positionnement d'électrodes en électroencéphalographie

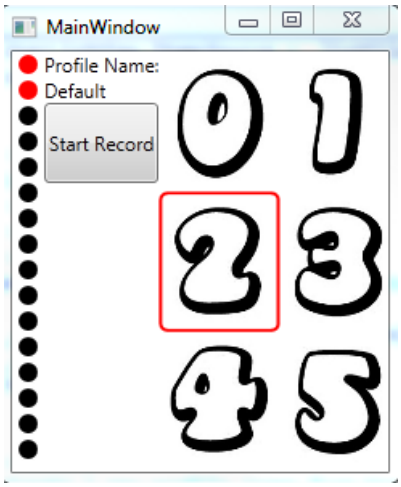

FIGURE 4.1 – Interface de SearchP300

doit avoir un repère temporel. Dans notre cas, nous nous baserons sur l'évènement attendu (flash du chiffre encerclé).

Version basé sur des timers

Une première version de cette application (SearchP300) insérait le champs "Flash" dans le fichier d'acquisition des données du casque au moment ou le flash attendu se produisait. Les acquisitions provenant des dictionnaires de la librairie emotivLib. Plusieurs problèmes sont cependant à dénoter :

Pour être synchronisé sur un flash, il nous faut savoir à quel moment précis le flash a eu lieu dans la base de temps du casque : Il faut déterminer sur quelle mesure ce flash a eu lieu. Or à l'instant où le flash a lieu, la donnée que nous voulons "marquer" est tout juste détecteé par le casque. Elle sera ensuite transmise dans le buffer, transmise à EmotivLib et enfin écrite dans le fichier.

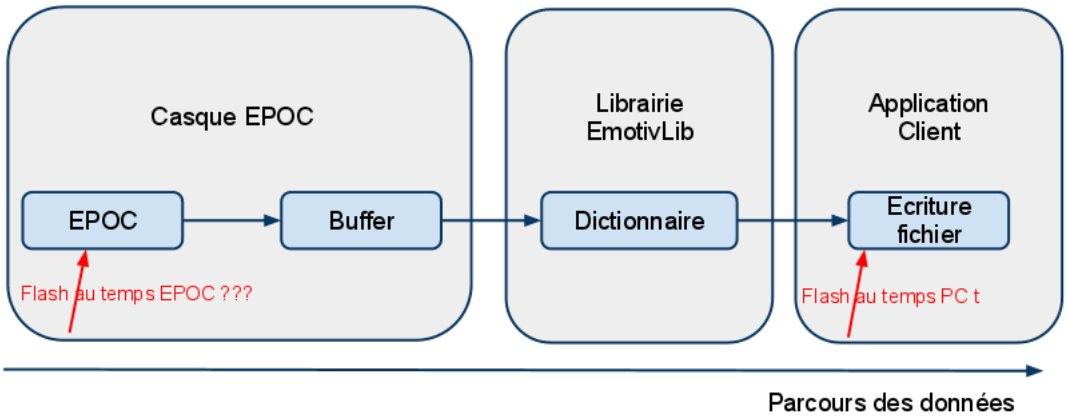

FIGURE 4.2 – Chemin de données

Le seul moyen de la retrouver est de calculer le temps de propagation depuis le casque jusqu'au fichier. Ceci peut être fait en établissant un point de synchronisation à un instant T avant toute capture entre le PC et le casque. On détectera alors la mesure sur laquelle le flash à eu lieu en calculant la différence de temps.

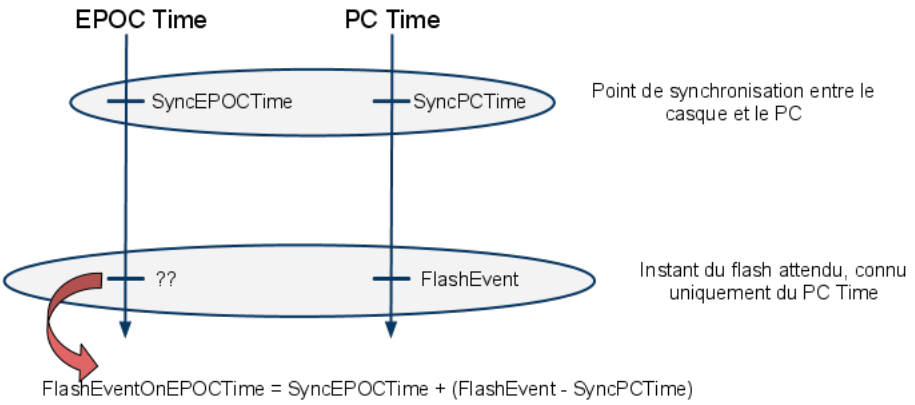

FIGURE 4.3 – Détection du temps EPOC du flash

Pourtant cette solution n'est pas sans problème puisque qu'il faut établir à un moment donné une synchronisation entre le casque et le PC. Puisque les données transitent toujours par la librairie et que le buffer contient généralement entre 12 et 16 lignes de données (échantillonnées à 125Hz), même lors de la synchronisation nous serons légèrement en retard. Cette synchronisation aura alors potentiellement une erreur allant jusqu'à environ 0.1 seconde (12*(1/125)).

Version sans emotivLib

Une deuxième version de l'application (ReSearchP300) a été réalisée sans la librairie emotivLib. Pour synchroniser plusieurs prises, c'est le thread qui récupère les buffers qui s'occupe aussi de flasher les chiffres. Tous les X buffers lus, nous envoyons un flash. Si le flash est le celui attendu, alors on démarre l'enregistrement des 15 prochains buffers dans un fichier de sortie. Ici le flash intervient quasiment toujours au même instant (même si le temps que met l'application WPF à afficher le flash peut varier d'une exécution à l'autre, ce temps très cours est négligeable). Nos données sont donc en théorie correctement synchronisées.

La difficulté de ces captures réside sur le fait que nous ne pouvons aucunement savoir si les données enregistrées sont effectivement bien synchronisées entre elles ou non. Nous pouvons nous baser que sur une idée théorique du problème. Or la détection du P300 devant se faire avec un post traitement sur ces données, si ces dernières sont erronées, nous ne risquons pas de détecter quoi que ce soit...

4.2.2 Acquisition via Emotiv TestBench

En plus de cette acquisition des données bruts que nous venons de détailler, nous avons aussi chercher à utiliser des outils existants pour rechercher le P300. Il existe un plugin MatLab nommé EEGLAB spécialisé dans l'analyse de données en électroencéphalographie. Or Emotiv permet l'exportation des données dans un format reconnu par EEGLAB (.edf). Cette exportation ne peut cependant pas se faire via le SDK, mais uniquement à l'aide du programme Emotiv TestBench.

La recherche d'ERP via EEGLAB nécessite d'inclure des évènements (`epoch`[2]), de la même manière que nous avons insérer le champ "Flash" dans les fichiers csv. Dans Emotiv TestBench, ces évènements

2. temps de référence

sont appelés `Marker`. Ils peuvent être insérés manuellement ou selon un signal provenant du port série.

Nous avons donc réalisé un programme `SearchP300TestBench` qui flash des chiffres comme pour l'application précédente et à chaque évènement attendu, un message est envoyé sur un port série virtuel. Pour l'émulation du port virtuel, nous utilisons Virtual Serial Ports Emulator de Eterlogic (VSPE). En mode `connector`, VSPE permet de créer un port virtuel ouvrable deux fois, ouvrant ainsi un tunnel entre deux applications clientes. Les paramètres utilisés pour cette liaison série sont les suivants :

- Port COM2
- 9600 Bauds
- Aucun bit de parité
- Stop bit à 1
- Pas de contrôle

5 Traitement des données

Nous expliquerons dans cette partie quels ont été les moyens mis en oeuvre pour essayer d'extraire un ERP des données capturées. Cette détection n'étant absolument pas garanti avec le `EPOC`, nous nous sommes tout d'abord concentrés sur l'utilisation d'outils externes spécialisés en électroencéphalographie. Puis en vue des résultats positifs, nous avons élaboré une librairie `python`.

5.1 EEGLAB

EEGLAB se présente comme un plugin MatLab spécialisé dans le traitement de données électrophysiologiques. En plus de sa version MatLab standard, ce produit open-source fournit aussi une version standalone. Etant donnée la complexité de l'outil, nous nous concentrerons uniquement sur les points nécessaire à la compréhension et à l'extraction des ERP.

5.1.1 Affichage d'un ERP

Une fois le plugin installé [1], on peut le configurer dans simplement en ajoutant le path

```
addpath('PathToEEGLAB')
```

Puis l'interface graphique se lance avec la commande `eeglab`.

Le programme permet alors de charger des données sauvegardées au format .edf, ce qui correspond au format exporté par Emotiv TestBench.

```
File -> Import data -> From EDF
```

Le plus simple consiste alors à charger tous les canaux, même si nous n'en utiliserons que quatre.

Normalement, dès le chargement du fichier .edf, toutes les informations nécessaire concernant le casque devraient être correctement chargées. Dans `[ Edit -> Dataset Info ]`, il faut s'assurer que la fréquence d'acquisition des données est bien de 128Hz.

On peut aussi saisir la position des capteurs sur le crane dans `[ Edit -> Channel locations ]`. Un fichier contenant les positions standards du EPOC est disponible. Cette étape permettra par exemple de visualiser les zones de l'activité cérébrale en fonction du temps. Elle n'est cependant pas nécessaire pour évaluer la présence d'un P300.

EEGLAB permet de sélectionner les évènements en fonction de leur nom `[ Edit -> Select data using events ]`. Dans SearchP300TestBench, on envoie toujours la même marque à chaque évènement attendu, on peut donc laisser la valeur par défaut et prendre en compte tous les évènements. Mais dans le cas ou nous aurions plusieurs évènements différents, on pourrait en choisir un et évaluer les ERP

1. http ://sccn.ucsd.edu/eeglab/

uniquement aux instants de cet évènement.

Nous pouvons alors extraire les `epochs`, en définissant l'intervalle sur lequel nous voulons analyser les données [Tools -> Extract epochs]. Les données seront traitées depuis epoch-start jusqu'à epoch+end. Toutes les données qui ne sont pas dans un intervalle ne sont pas analysées. Le schéma ci-dessous permet de visualiser ce que fait EEGLAB lorsque l'on extrait les évènements des canaux.

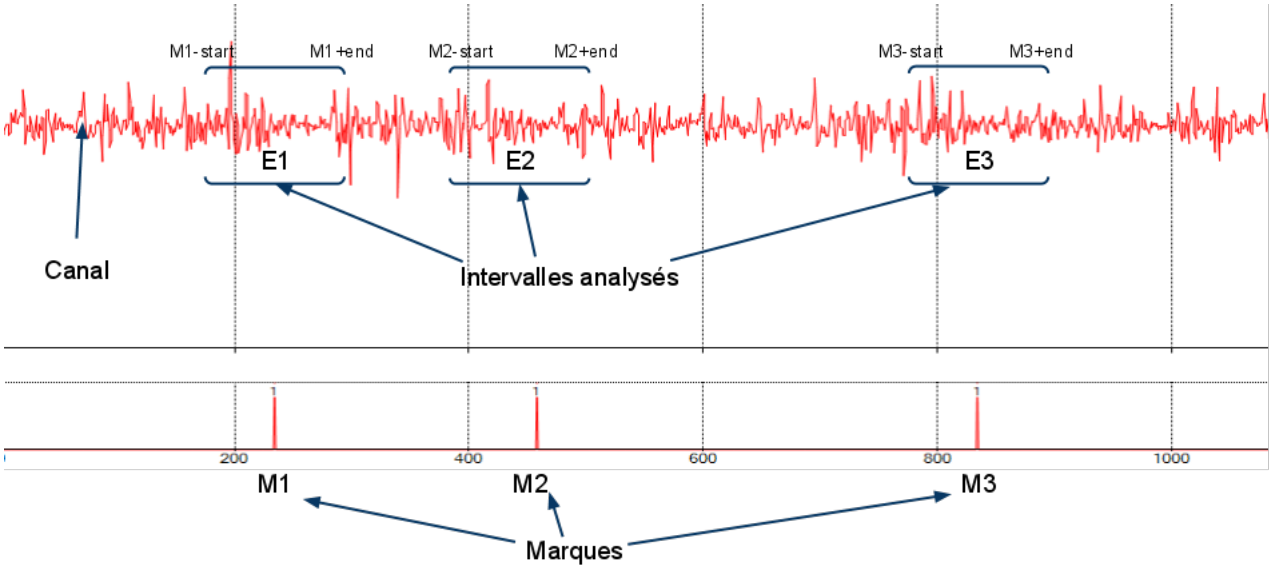

FIGURE 5.1 – Découpage des données selon un epoch

Une fois que le découpage à été effectué, les données sont alignées verticalement pour être traitées.

L'affichage de l'ERP se fait avec `[ Plot -> Channel ERP image ]`. Il faut alors choisir quel est le capteur que nous voulons afficher. Puisque nous avons préalablement chargé tous les canaux, il faudra utiliser les correspondances suivantes :

Capteur	Channel Number
AF3	3
F3	5
F4	14
AF4	16

5.1.2 Analyse des résultats

Plusieurs séries de mesures ont été effectuées par 5 individus. Le casque fut placé en arrière, de façon à rapprocher les capteurs F3 et F4 du sommet du crane. Les données furent enregistrées via SearchP300TestBench. Pour deux des cinq individus, nous n'avons pu discerner aucun résultat significatif. Les données s'apparentaient plus à du bruit qu'à des signaux exploitables.

Mais pour les trois autres nous avons pu observer pics positifs de tension d'environ 5 micro volts sur les capteurs F3 et F4 (Parfois sur un seul des deux). Ces pics interviennent environ 300 millisecondes après l'epoch, parfois un peu plus tard. Cependant ces résultats sont très dépendants du placement du casque. Il arrive que lors d'une même session d'enregistrement, l'on arrive à ces résultats une première

fois, puis plus rien 30 minutes plus tard alors que le casque semble placé dans la même position.

Sur les figures suivantes, chaque ligne de couleur représente un essai. La couleur représente la tension du signal. L'évènement attendu correspond au temps T=0. Le graphique du dessous est défini par le moyenne des essais.

Signalons que les données affichées ne correspondent pas directement aux données capturées brut, elles ont préalablement été traitées par EEGLAB pour mettre en évidence un éventuel ERP. Nous verrons dans le chapitre suivant quels traitements ont été appliqués pour obtenir ces résultats.

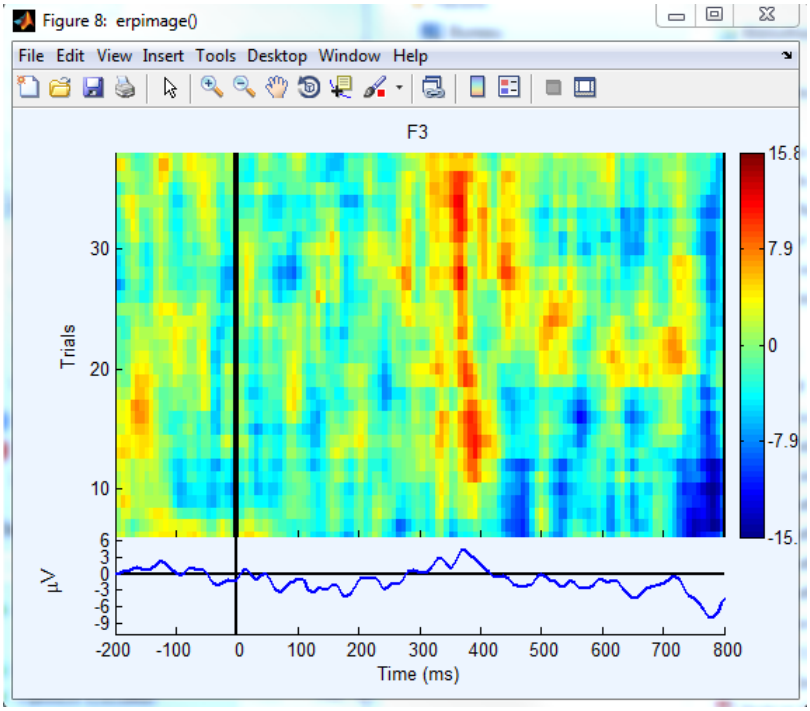

FIGURE 5.2 – Capteur F3 sur 40 essais

5.2 Librairie PlotEPOC.py

Suite aux résultats encourageants obtenu via EEGLAB, nous avons mis en place une librairie Python qui s'occupe de l'analyse des données enregistrées. Cela permettra de ne plus être dépendant du logiciel TestBench, ni même de EEGLAB. Nous pourrons alors obtenir une application complète et indépendante pour le traitement des données.
Ce chapitre détaillera quelques moyens pis en oeuvre pour extraire un ERP à partir des fichiers .csv préalablement capturés.

5.2.1 Utilisation standard

PlotEPOC.py permet de charger, d'interpréter et d'afficher les données capturées par le EPOC. Puisque nous avons mis en place deux méthodes différentes de capture des données brut, la librairie permettra de charger les deux formats. Les données capturées avec `SearchP300` sont contenues dans un unique fichier .csv. Pour procéder à son chargement, il faut passer ce fichier en argument.

```
python monFichier.csv
```

Quant aux données capturées avec `ReSearchP300`, les mesures de chaque évènement attendu sont stockés dans des fichiers séparés. Pour les charger, il faut mettre toutes les mesures dans un même dossier et passer ce dernier en argument.

```
python monDossierContenantTousLesCSV/
```

5.2.2 Fonctionnement

La librairie comporte une classe PlotEPOC. Pour créer une instance, seul le path est un paramètre indispensable.

```
def __init__(self, path, nbrOfLinePerEPOCH=128, movingAverageSize=10, resX=640, resY
    =640):
```

Le paramètre `nbrOfLinePerEPOCH` définit combien de mesures seront traitées après l'évènement attendu (les données étant capturées à une fréquence de 128Hz, la valeur par défaut correspond donc à une seconde). Le paramètre `movingAverageSize` définit la largeur de la moyenne glissante appliquée aux données. `resX` et `resY` correspondent respectivement à la largeur et la hauteur de la fenêtre d'affichage en pixel.

Les données extraites sont alignées verticalement (voir figure 5.1), synchronisées sur l'évènement attendu, et sont stockées dans un attribut de type dictionnaire avec comme clés les noms des quatre capteurs utilisées.

```
self.data =
{
    'AF3': [[],[],[],...],
    'F3': [[],[],[],...],
    'F4': [[],[],[],...],
    'AF4': [[],[],[],...] }
}
```

Chaque élément du dictionnaire est alors composé d'une liste contenant les évènements attendus. Chacun de ces évènements est alors une liste des mesures capturées. Il y a `nbrOfLinePerEPOCH` mesures par évènement.

Suppression des mauvais essais

Le premier traitement appliqué aux données consiste a supprimer les essais erronés, en particulier les pics de tensions positifs ou négatifs due aux mouvements de l'utilisateur. La sélection se base sur le nombre d'essai ayant plus de N valeurs n'appartenant pas à un intervalle donnée. Cet intervalle est défini en fonction de l'écart type de toutes les mesures du canal. Par défaut, on gardera uniquement les essais qui ont moins de 20 valeurs dépassant la moyenne plus ou moins 4 fois l'écart type. Ces variables peuvent être modifiées dans la formule suivante :

```
filter(lambda elem : len(filter(lambda d : abs(d-(mean+std))>(4*std), elem))<20, data)
```

Ce premier traitement améliore les analyses ultérieurs, mais il n'est toutefois pas indispensable. Si trop de données sont supprimées et que les paramètres sont difficile à régler pour un utilisateur, il peut être bypassé sans compromettre la suite du traitement.

Lissage est essais

Pour extraire l'ERP, nous appliquons ensuite une moyenne glissante entre les essais. Contrairement à l'étape précédente, celle ci est indispensable pour pouvoir visualiser le P300. En effet, ce traitement permet de considérablement augmenter le ratio signal / bruit. Plus le nombre de mesures capturés par l'utilisateur est grand, plus la réduction du bruit sera efficace avec cette méthode.

Cette moyenne glissante est effectuée verticalement (entre les essais), pour chaque instant capturé (moyenne glissante entre les essais au temps t0, puis au temps t0+periode, etc...). Ce traitement nécessite donc que les données capturées soient synchronisées entre elles (voir le chapitre 4.2 pour les mécanismes d'acquisitions), sinon toutes les mesures se retrouveraient faussées.

La largeur de la moyenne glissante vaut 10 par défaut. Cela implique qu'il y ait au minimum 10 essais enregistrés. Si les données sont très bonnes, on peut essayer de réduire cette largeur, mais en règle générale, une valeur comprise entre 5 et 12 est appropriée.

Affichage des essais

Afin de déterminer si un ERP est présent ou non dans les données traitées, nous affichons les essais de la même manière que EEGLAB. L'affichage a été réalisé avec pygame, un wrapper de la librairie SDL en python. C'est la méthode `plotMultipleTry()` qui gère l'affichage.

Le choix de la couleur de chaque donnée capturée sera fonction de son positionnement par rapport à l'écart type. Nous utilisons un dégradé passant par les valeurs suivantes :

- plus de 4 fois l'écart type à la moyenne : rouge foncé
- valeur moyenne : Vert
- moins de 4 fois l'écart type à la moyenne : bleu foncé

Ces valeurs peuvent être modifiées à l'appel de la fonction `__getColor(valeur, minimum, maximum)` qui s'occupe de transformer la valeur passé en paramètre en RGB, en fonction de sa position dans l'intervalle `minimum` à `maximum`.

En dessous des essais, un graphique représente une moyenne des essais, toujours synchronisés temporellement.

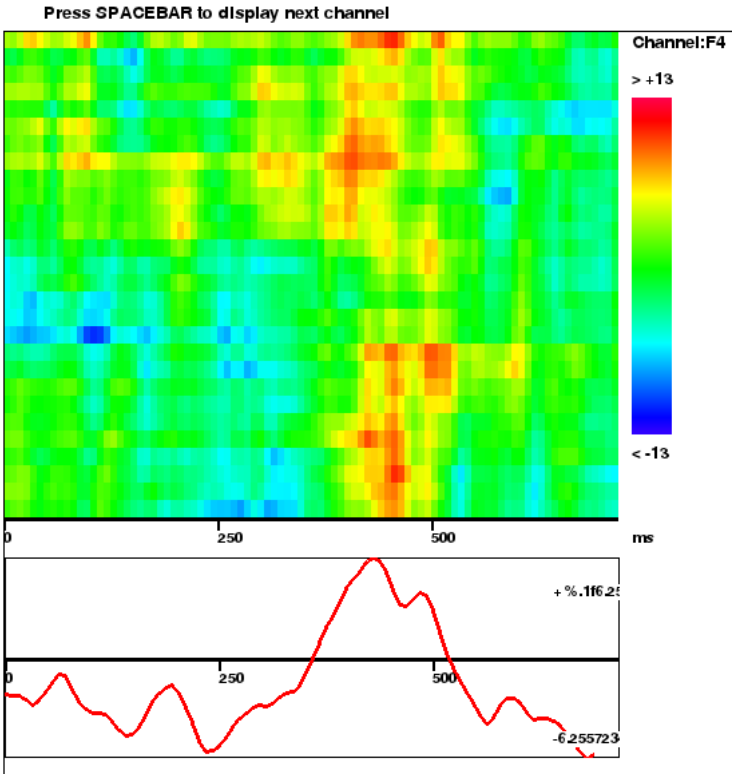

FIGURE 5.3 – Affichage de PlotEPOC