TL SIR
Commande de robot par interface cérébrale, par gestes ou par la parole

Béatrice Chevaillier Jean-Luc Collette Jean-Louis Gutzwiller
Michel Ianotto Stéphane Rossignol Jean-Baptiste Tavernier

12 octobre 2015

Table des matières

1 Introduction

1.1 Objectif de ce TL

L'objectif de ce TL sur 14 séances est de mettre en pratique et d'approfondir un certain nombre de méthodes abordées dans différents cours de la majeure SIR. L'application proposée est de commander un robot Koala soit par la parole, soit par des gestes, soit grâce aux signaux issus d'une interface cérébrale (*brain computer interface* ou BCI). Pour des questions de temps, **vous n'en étudierez que certains aspects** qui varieront suivant le sujet que vous choisirez parmi les trois proposés ci-dessous.

Pour la répartition, nous accepterons au maximum 3 binômes pour la commande par la parole, 3 pour celle par gestes et 2 pour celle par BCI.

1.2 Principes communs

Pour les trois modalités, un système permet d'obtenir des signaux (son, image, signaux cérébraux) en temps réel. Le travail peut se décomposer en deux grandes étapes :

— une extraction d'attributs sur les signaux, afin de déterminer l'ordre émis par l'utilisateur ;
— le contrôle du robot, qui est indépendant de la modalité choisie.

Pour la commande du robot, on se limitera aux ordres suivants : avancer, tourner à gauche, tourner à droite. Pour la commande par les signaux cérébraux, le robot devra rester immobile en l'absence d'ordre ; pour la commande par la parole ou par geste, en revanche, un mot ou un signe spécifique correspondra à l'arrêt du robot.

Nous vous conseillons de parcourir d'abord l'énoncé dans son intégralité pour avoir une vue d'ensemble du travail demandé, des méthodes proposées et des liens entre les différentes « briques » que vous élaborerez : vous pourrez en effet être amenés à mettre en œuvre plusieurs programmes à la suite les uns des autres, et tous les détails ne figurent pas nécessairement dans l'énoncé, à vous de réfléchir ! De nombreux éléments sont à votre disposition en annexe de ce TL.

Les différentes modalités de commande sont détaillées dans la partie 2. Les moyens de commande du robot sont expliqués dans la partie 3.

1.3 Travail à effectuer

— Simulation pour l'extraction d'attribut pour une modalité de commande au choix (Matlab ou Octave).
— Ecriture d'un programme en C ou C++ qui fournit en sortie les ordres au robot (avancer, tourner à droite, tourner à gauche, s'arrêter).
— Ecriture d'un programme en C ou C++ qui, à partir des ordres, contrôle le robot.

Pour le contrôle du robot, il y a trois modalités possibles : il est demandé d'en mettre en œuvre au moins une, mais d'autres pourront être testées en fonction du temps disponible.

1.4 Délivrables

A la fin des 14 séances, vous remettrez un rapport sur votre travail, et vous nous en présenterez oralement le contenu lors d'une soutenance d'environ 30 minutes. Nous attendons de vous des explications sur la manière de mener vos tests, des conclusions sur la validité des méthodes, le choix des paramètres... même si ce n'est pas explicitement mentionné au fil de l'énoncé.

2 Modalités de commande

2.1 Extraction des ordres à partir des signaux de parole

2.1.1 La tâche

Un micro permet d'enregistrer les sons : le package `supelec-audio` (http://www.metz.supelec.fr/metz/personnel/gutzwiller/_TUTOS_98763456_/20100908/Supelec-audio/Supelec-audio.pdf)permet de récupérer les échantillons dans votre programme. Le programme devra reconnaître quelques mots/courtes phrases. Nous vous suggérons d'utiliser par exemple : EN AVANT, À DROITE, À GAUCHE, STOP. Il faudra ensuite envoyer les ordres au robot. Les ordres devront être émis selon le protocole décrit dans la partie 3. Auparavant, il est nécessaire de tester les différentes étapes et de définir les limites des techniques proposées.

2.1.2 Reconnaissance de la parole

Les outils de traitement de la parole se décomposent communément en deux parties : l'extraction de caractéristiques et le décodage. Voir la figure 1 (sur cette figure, nous montrons l'extraction de l'énergie et des MFCC, ainsi que de leurs dérivées première Δ et seconde Δ^2 : au cours de ce TL, nous nous intéresserons aux LPC et nous nous contenterons, au moins dans un premier temps, des MFCC sans leurs dérivées ni l'énergie).

La première étape consiste à extraire des caractéristiques, via des techniques de traitement de signal appliquées à des trames de son de quelques dizaines de millisecondes. Ces caractéristiques doivent mettre le mieux possible en évidence les particularités des différents segments de la parole. Ces segments peuvent être considérés à des niveaux variés : par exemple, les phonèmes ([a], [e], [i], [t], [s], etc. ; ce qui nous amène à considérer un problème à une trentaine de classes), ou les phonèmes contextualisés ([am], [an], [al], [bl], etc. ; ce qui nous conduit à un problème de plusieurs milliers de classes), ou etc. Les caractéristiques les plus étudiées sont les coefficients de prédiction linéaire (LPC) et les mèl filter cepstrum coefficients (MFCC). Nous allons les aborder en détails au cours de ce TL.

La deuxième étape se décompose en deux sous-étapes. Premièrement, il s'agit de classifier chacune des trames du son. Comme le nombre de classes est très important, il faut être capable de donner une liste de N meilleures hypothèses. Deuxièmement, ces N meilleures hypothèses sont élaguées, en ajoutant des contraintes telles que les successions possibles de phonèmes, les mots possibles, etc., ce qui permet finalement de reconstruire les mots qui ont/auraient été dits (notez qu'une étape suivante serait d'interpréter ce

qui a été dit : nous n'allons pas considérer ceci ici). L'état de l'art indique que le couple GMM (Gaussian Mixture Model) pour le classifieur, et HMM (Hidden Markov Model) pour l'élagage, est ce qui est communément utilisé. Cependant, ici, la reprogrammation de l'ensemble GMM-HMM n'étant pas envisageable, nous allons considérer la DTW (Dynamic Time Warping), qui est une méthode plus aisée à implémenter et qui accomplit les deux sous-étapes simultanément. Cependant, cette méthode ne permettra pas l'utilisation de phrases trop complexes pour les ordres à transmettre au robot.

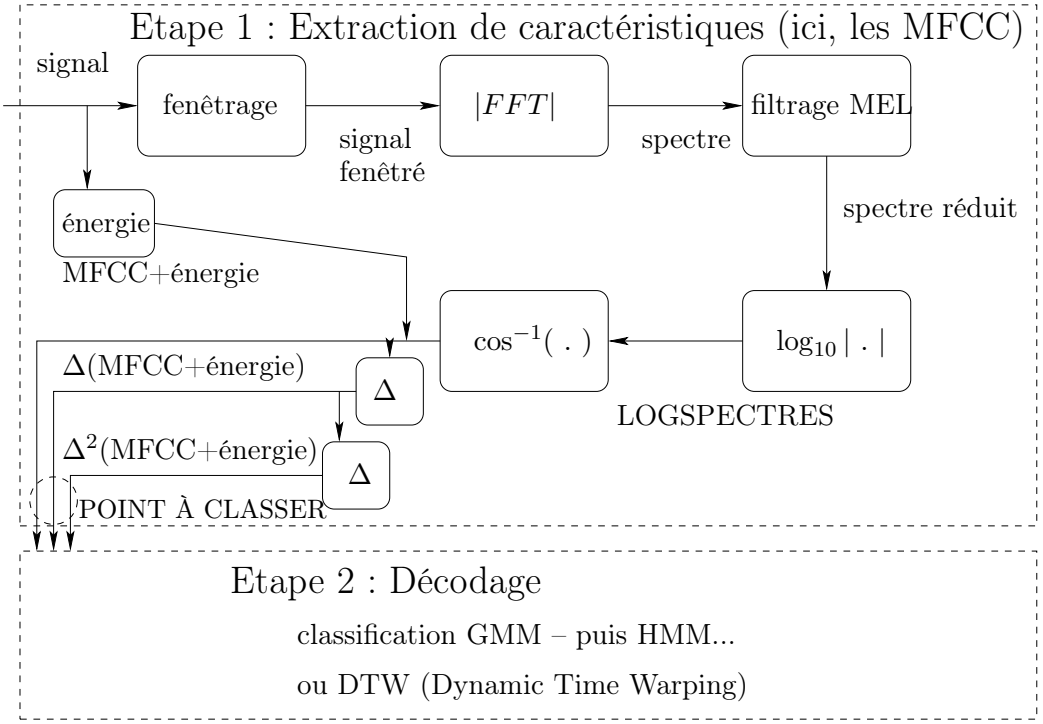


FIGURE 1 – Les modules

2.1.3 Principaux thèmes abordés

- Acquisition des sons
- Caractérisation des sons (dans des conditions stables : mono-locuteur, pas ou peu de bruit... ; puis dans des cas plus généraux : plusieurs locuteurs, ...)
- Décodage par la DTW

2.1.4 Pré-traitement et extraction de caractéristiques

Généralités Comme indiqué, deux types de de caractéristiques vont être étudiées : les coefficients de la prédiction linéaire (voir l'annexe 4.4) et les MFCC (voir l'annexe 4.2).

Les tests et ajustements des paramètres seront d'abord réalisés sous un logiciel de calcul matriciel (Matlab ou Octave), avec des sons pré-enregistrés par nous puis par vous ; puis les techniques seront implémentées en C et/ou C++ et testées hors-ligne.

Remarque sur les tests Nous vous demandons, lors de la validation de cette étape, de contribuer à élargir notre base de données en sauvegardant vos sons.

2.1.5 Décodage

Toutes les productions mono-locuteur d'un même son (par exemple, un [a] soutenu), dans les mêmes conditions d'enregistrement, sont différentes, déjà d'un point de vue du contenu spectral. Les caractéristiques permettent normalement (dans le cas idéal) de s'affranchir de ces différences, rendant possible de ce fait une classification efficace. Malheureusement, ces productions diffèrent aussi en longueur ! Par exemple, un locuteur donné ne dit jamais le [o] de STOP exactement avec la même longueur. Pour des mots complets ou des phrases courtes complètes, ce phénomène est encore plus net. La classification n'est donc pas suffisante. Quand on veut comparer deux phrases dites par un locuteur pour décider si la même chose a été dite, il faut les recaler en temps. La DTW est utilisée pour ce faire (voir l'annexe 4.3).

Pour faire les tests, nous vous suggérons d'utiliser les sons que nous vous fournissons, en prenant comme sons de référence par exemple l'ensemble *agauche.wav*, *adroite.wav*, *enavant.wav* et *stop.wav*, et comme sons à reconnaître l'ensemble *agauche2.wav*, *adroite2.wav*, *enavant2.wav* et *stop2.wav*.

Il faudra ensuite enregistrer vos propres sons, et tester de nouveau les techniques.

Les tests et ajustements des paramètres seront d'abord réalisés sous un logiciel de calcul matriciel (Matlab ou Octave), avec des signaux artificiels, et avec des sons pré-enregistrés par nous puis par vous ; puis les techniques seront implémentées en C et/ou C++ et testées hors-ligne.

2.1.6 Remarque pour le temps-réel

Finalement, il faudra passer en temps-réel, c'est-à-dire commander le robot en temps réel, et tester.

Quand vous accomplirez l'implémentation en temps-réel, il faudra penser à ajouter un module de détection de silence, afin de déterminer les parties du son à analyser. Cette détection de silence pourra être effectuée grâce à l'analyse de l'énergie dans un premier temps, ou à l'aide de méthodes plus sophistiquées si ça ne suffit pas.

2.2 Extraction des ordres à partir des gestes

Dans la chaîne de traitement complète, une webcam filme les gestes de l'opérateur. Le programme devra reconnaître les gestes et émettre les ordres vers le robot. Il faut donc d'abord analyser l'image avant d'en déduire les ordres à envoyer. Dans l'étape finale de réalisation, on utilisera `ROS` pour communiquer avec la webcam et le robot. Auparavant, il est nécessaire de tester les différentes étapes et de définir les limites de la méthode. Dans cette phase de test, on pourra alors réaliser l'acquisition des images directement avec `OPENCV`.

Le traitement peut se décomposer en deux grandes parties :

- extraction d'attributs caractéristiques du signe utilisé sur une image,
- reconnaissance du signe grâce à ces attributs.

2.2.1 Principaux thèmes abordés

- Acquisition d'images avec `OPENCV` puis `ROS`.
- Caractérisation d'une couleur indépendamment de la luminosité.
- Segmentation d'images.
- Suivi de contour sur une image binaire.
- Caractérisation d'une forme par descripteurs de Fourier ; construction d'un vecteur d'attributs.
- Apprentissage non supervisé : quantification vectorielle par K-moyennes.
- Apprentissage supervisé : K plus proches voisins, réseaux de neurones...

2.2.2 Constitution de la base de gestes

Nous vous suggérons d'utiliser les gestes de la figure 2 pour commander le robot. Le point fermé indique au robot de s'arrêter, la main ouverte avec les doigts écartés informe le robot qu'il doit avancer, la main ouverte avec les doigts serrés indique que le robot doit tourner à droite ou à gauche selon l'orientation de la main.

A l'aide d'un logiciel d'acquisition vidéo (`cheese` ou `mencoder` par exemple), vous réaliserez l'acquisition des différentes séquences vidéos correspondant aux gestes à reconnaître. Chaque catégorie de gestes sera représentée par au moins 200 images. Une classe constituée d'images de rejet sera ajoutée aux classes précédentes.

2.2.3 Pré-traitement et extraction d'attributs

Il s'agit d'abord de segmenter la main puis d'extraire son contour ordonné. Nous vous proposons de caractériser ce contour par ses descripteurs de Fourier, qui constitueront le vecteur d'attributs utilisé pour la reconnaissance de forme. Ces traitements seront réalisés en `C++`, en faisant appel à la librairie `OPENCV`. Dans une phase de test et avant d'utiliser `ROS`, on pourra faire appel pour l'acquisition ou la sauvegarde des images ou des vidéos, aux fonctions décrites dans la page `http://docs.opencv.org/modules/highgui/doc/reading_and_writing_images_and_video.html`, en particulier le mode d'emploi de la fonction `VideoCapture` qui permet de lire un flux vidéo à partir d'un fichier ou directement de la webcam.

Le programme pourra ainsi traiter aussi bien les séquences vidéos de la base de gestes que le flux en provenance de la webcam. Pour un programme `traitement.cpp`, la ligne de commande permettant de générer le programme exécutable `traitement` est donnée ci-dessous.

```
g++ traitement.cpp `pkg-config -cflags -libs opencv` -o traitement
```

Dans la phase finale de réalisation, on utilisera `ROS` pour gérer l'acquisition des images.

Détection de couleur Il s'agit de mettre en œuvre une détection suffisamment robuste pour être relativement indépendante des conditions de prise de vue. On peut par exemple envisager de sélectionner les points caractérisés par leur composante couleur (R, G, B) vérifiant :

- $\max(R, G, B) - \min(R, G, B) \geq seuil$
- $\max(R, G, B) = R$.

(a)

(b)

(c)

FIGURE 2 – Les trois gestes à reconnaître. (a) : le robot s'arrête, (b) : le robot avance, (c) : le robot tourne à droite ou à gauche selon l'inclinaison de la main.

Ainsi, on élimine les points pour lesquels la couleur est peu saturée (proche des teintes de gris), et qui ont une composante dominante dans le rouge (comme la main).

Suivi de contour On pourra utiliser la fonction `findContours` de `OPENCV` pour réaliser ce suivi. Vous trouverez la documentation sur `http://docs.opencv.org/`. On choisira l'option `CV_RETR_EXTERNAL` pour limiter le nombre de contours extraits, et l'option `CV_CHAIN_APPROX_NONE` pour obtenir la liste complète des points des contours. Tous les contours sont détectés et on peut considérer que celui de plus grande longueur représente la main.

Evaluation des descripteurs de Fourier Les descripteurs de Fourier d'un contour permettent de construire un vecteur d'attributs invariant par translation, rotation et changement d'échelle (homothétie) de ce contour, pouvant être utilisé pour de la clas-

sification. Vous trouverez quelques éléments sur les descripteurs de Fourier sur la page http://sirien.metz.supelec.fr/spip.php?article77.

Il est conseillé, entre autres choses, d'essayer de reconstruire un contour à partir d'un nombre réduit de descripteurs, et d'observer les différences avec le contour initial. Pensez à cette étape à faire un choix judicieux pour les signes de commande...

Remarque sur les tests Nous vous demandons, lors de la validation de cette étape, de contribuer à élargir notre base de données en enregistrant des films avec une succession de signes et en testant vos programmes dessus.

2.2.4 Classification

Il s'agit maintenant de reconnaître les signes grâce aux vecteurs d'attributs. Nous proposons de tester plusieurs méthodes.

— Apprentissage non supervisé : quantification vectorielle par K-moyennes.
— Apprentissage supervisé : K plus proches voisins, réseaux de neurones.

K-moyennes Il s'agit d'un algorithme de quantification vectorielle qui est également utilisé pour l'apprentissage non supervisé. Ce n'est pas à proprement parler un algorithme de classification, puisqu'il faut une étape supplémentaire pour étiqueter les classes.

Soit $(\xi_i)_{1\leq i\leq N}$ un ensemble de vecteurs dont les composantes sont les descripteurs de Fourier de N contours $(C_i)_{1\leq i\leq N}$. L'algorithme recherche les K prototypes $\{\mathbf{w}_1,\ldots,\mathbf{w}_K\}$ qui minimisent la fonction de coût :

$$\widehat{\Psi}(\mathbf{W}) = \frac{1}{N}\sum_{j=1}^{K}\sum_{\xi_i\in V_j}\|\xi_i-\mathbf{w}_j\|^2, \tag{1}$$

où $\mathbf{W}=[\mathbf{w}_1,\ldots,\mathbf{w}_K]$ est la matrice dont les colonnes sont les vecteurs prototypes $\mathbf{w}_j$ et où V_j est le polyhèdre de Voronoï de $\mathbf{w}_j$. Le principal défaut de l'algorithme des K-moyennes est que le résultat obtenu est très dépendant des conditions initiales et peut correspondre à un minimum local et non global de la distorsion [2] ; en pratique, il suffit cependant la plupart du temps de lancer l'algorithme avec plusieurs conditions initiales différentes pour résoudre ce problème. Cf. annexe 4.5 pour plus de détails.

Pour faire les tests, nous vous suggérons de lancer l'algorithme en mettant en entrée un ensemble de vecteurs attributs correspondant à différents signes, avec des classes équilibrées ou non, et de vérifier *a posteriori* si les vecteurs représentant les mêmes signes se retrouvent dans une même classe ou pas.

Il est alors possible, par quantification vectorielle, d'attribuer tout nouveau vecteur à l'un des représentants trouvés précédemment, donc reconnaître le signe correspondant.

K plus proches voisins Il s'agit d'un algorithme d'apprentissage supervisé, qui nécessite donc une base de données étiquetée. Pour classer un contour décrit par le vecteur ξ, on cherche les K vecteurs de la base d'apprentissage qui sont les plus proches de ξ : on attribue à celui-ci l'étiquette majoritairement présente dans ses K plus proche voisins. Il existe des algorithmes qui accélèrent le processus en évitant le calcul de l'ensemble des distances. L'algorithme de base est donné en annexe 4.6.

Les réseaux de neurones Il s'agit d'utiliser des algorithmes d'apprentissage basés sur les réseaux de neurones tels que le perceptron multicouches ou le réseau RBF. Ces algorithmes ont été vus dans le cours d'apprentissage automatique et sont décrits en détail dans le polycopié du cours. On rappelle en annexe 4.7 l'algorithme d'apprentissage du perceptron multicouches.

2.2.5 Utilisation de ROS

Dans l'étape finale de réalisation, on utilisera ROS pour réaliser les différentes interfaces, en s'inspirant des informations disponibles sur http://sirien.metz.supelec.fr/depot/SIR/TutoROS/cv.html.

2.3 Extraction des ordres à partir des signaux de signaux cérébraux

Un système d'acquisition cérébral permet d'obtenir des signaux en temps réel depuis une personne portant le dispositif. Dans le cadre de ce TL, les signaux sont disponibles sous forme d'enregistrements et peuvent être rejoués à volonté (voir 4.1.6 page 15). Dans la mesure du temps disponible (et s'il y a un volontaire pour porter le système d'acquisition), il sera possible de tenter l'expérience en vraie grandeur. Une fois l'ordre reconnu, il faut le transmettre au robot. Les ordres devront être émis selon le protocole décrit au paragraphe 3.

2.3.1 Principe de fonctionnement

Vous trouverez une introduction aux *steady-state visual evoked potentials* (SSVEP) dans [3]. Sur la console de commande sont fixées trois plaques clignotantes. Chacune de ces plaques clignote sur une fréquence particulière (actuellement $7,5$; 11 ; $13,5$ Hz) qui correspond à un ordre donné. Lorsque l'expérimentateur regarde l'une de ces plaques, la fréquence correspondante est présente dans son électroencéphalogramme. Il est demandé d'écrire un programme « analyse_BCI » qui analyse les échantillons issus du capteur BCI et qui les traduit en ordres pour le robot. Avant de passer à l'implémentation temps réel, il vous est demandé de réaliser une étude théorique complète de l'algorithme d'analyse des échantillons décrit ci-dessous, en vous aidant d'un logiciel de calcul matriciel (Matlab ou Octave) et en réalisant vos tests sur les signaux BCI fournis.

2.3.2 Etude théorique de l'algorithme d'analyse

Il s'agit de voir si l'utilisateur donne un ordre ou non et, dans le premier cas, de décider quel est cet ordre.

Distinguer les trois ordres On essaie de d'abord, sous l'hypothèse que l'utilisateur donne bien un ordre, de distinguer les trois commandes possibles : avancer, tourner à gauche ou tourner à droite.

Pour savoir laquelle des trois fréquences est présente dans le signal, il est souhaitable d'observer l'évolution au cours du temps de la puissance du signal autour de ces fréquences.

On commence donc par filtrer un signal d'entrée $x(n)$, qui est la moyenne des voies enregistrées sur la BCI, par un banc de filtres passe-bande centrés sur les fréquences précédentes, et synthétisés selon la méthode exposée dans l'annexe 4.8.

Il faut ensuite estimer la puissance des signaux en sortie du banc de filtres. Si on note y_i l'un de ces signaux, z_i défini par

$$z_i(n)^2 = (1-\alpha)y_i(n)^2 + \alpha z_i(n-1)^2 \tag{2}$$

est une estimation lissée de la puissance de y_i. Avec une valeur de α qui se rapproche de 1 par valeur inférieure, on obtient une estimation de plus en plus lissée.

Repérer l'absence d'ordre On suggère de choisir des fréquences non occupées par les signaux émis par les panneaux mais dans la bande qui nous intéresse, afin de caractériser ce qu'on pourrait considérer comme une activité cérébrale normale en l'absence des panneaux. On peut alors estimer un rapport signal à bruit pour chacun des signaux y_i et considérer qu'en deçà d'un certain seuil, aucun ordre n'est émis.

2.3.3 Programme d'analyse du signal BCI

Le programme « analyse_BCI », que vous écrirez, devra recevoir depuis un publiant ROS les signaux BCI issus soit d'un enregistrement, soit d'un capteur BIOSEMI. La fréquence d'échantillonnage est transmise par la source et est de 256 échantillons/seconde pour les enregistrements ou 2048 échantillons/seconde pour le capteur.

Le signal BCI étant issu d'un capteur multi-canaux, on commencera par fabriquer un signal mono-canal qui est la moyenne de tous les canaux avant d'appliquer le traitement du paragraphe précédent (expliquer l'intérêt de ce calcul de moyenne en ce qui concerne le bruit.

On pourra télécharger depuis http://ims.metz.supelec.fr/wiki/Depot/Gutzwiller/SIR/TL/Robots/acquisition_biosemi.zip un projet ROS contenant les deux programmes permettant d'effectuer la publication des données (« biosemi_talker » pour l'acquisition directe en provenance du capteur, ou « biosemi_simul » pour les données pré-enregistrées). Dans le cas de l'acquistion en directe des données du capteurs, le programme « acquisition_biosemi » devra être lancé sur la machine (sous Windows) à laquelle le capteur BIOSEMI est connecté.

Le programme « biosemi_listener » est un exemple de client recevant les données qui pourra vous inspirer pour écrire votre programme. La description complète de l'interface ROS se trouve en 4.1.6 page 15.

3 Contrôle du robot

3.1 Introduction

Ce programme devra, à partir des ordres reçus du programme précédent, commander le robot de sorte qu'il effectue bien les déplacements demandés. Afin que les déplacements du robot soient fluides, il faudra utiliser un modèle de commande adapté. Les étapes sont les suivantes :

— compréhension de la documentation de l'interface du robot à l'ordinateur ;
— traduction de la décision ci-dessus en commandes à envoyer au robot
— mise en œuvre d'un modèle de filtrage du comportement du robot pour le rendre plus fluide.

3.1.1 Ordres

En sortie des programmes d'analyse (suite à l'étape d'extraction d'ordres) seront fournies les commandes sous forme texte.
— Forward : le robot avance tout droit (sans tourner) ;
— TurnLeft : le robot tourne à gauche ;
— TurnRight : le robot tourne à droite ;
— None : ne rien faire (le robot s'arrête).
Le principe consiste à émettre régulièrement ces commandes à une fréquence fixe (par exemple 4 fois par seconde) tant que la condition est détectée. Chaque ordre doit être suivi d'un retour à la ligne.

3.1.2 Programme de commande

Le programme « commande_koala » devra
— prendre en entrée les commandes issues du programme d'analyse,
— les traduire en une vitesse pour chacune des roues (« VITESSE_ROBOT », « − VITESSE_ROBOT » ou « 0 »).
— fournir en sortie les informations à destination du robot (selon la modalité choisie).

3.1.3 Adaptation du programme au robot

Les robots dont vous disposez sont de deux types qui ont des caractéristiques différentes. Afin de faire fonctionner le programme de la même manière sur les deux robots, proposez une solution pour tenir compte des caractéristiques spécifiques de chacun des robots.

3.1.4 Rendre le mouvement plus fluide

Pour rendre le mouvement plus fluide, nous proposons de filtrer les vitesses des roues par un filtre passe-bas moyenneur avant de les envoyer au robot.

3.2 Les possibilités de contrôler le robot

3.2.1 Contrôle avec port série

Ce moyen de contrôle utilise un câble série connecté entre l'ordinateur de contrôle et la base roulante du robot. Pour utiliser ce moyen, il faudra débrancher le cordon série qui va de la base roulante vers le petit ordinateur de contrôle qui se trouve au-dessus de la base roulante. On otera la fiche de la prise de l'ordinateur de contrôle et on branchera cette fiche sur le câble relié à votre ordinateur.

Le nom du port série de la machine relié au robot koala doit être passé en paramètre. Ce nom dépend du type de machine.

— Sur Windows : les ports séries s'appellent habituellement « COM1 », « COM2 »... . Dans le cas de l'utilisation d'un adaptateur USB-série (pour les machines qui ne disposeraient plus de port série), les noms habituels utilisent des numéros plus élevés comme « COM7 ».
— Sur Linux : si la machine possède des cartes séries physiques, les noms sont habituellement « /dev/ttyS0 », « /dev/ttyS1 »... Dans le cas de l'utilisation d'un adaptateur USB-série, le port série prendra pour nom « /dev/ttyUSB0 »...

Le nom indiqué doit être utilisé en paramètre des fonctions mises à votre disposition au paragraphe 4.1.1. La documentation du robot, accessible depuis http://sirien.metz.supelec.fr/spip.php?article6, décrit le protocole à utiliser.

3.2.2 Contrôle en flux réseau

Ce moyen de contrôle utilise le réseau pour la communication entre votre application et le robot. Le cordon série de la base roulante doit alors être relié au petit ordinateur de contrôle qui se trouve au-dessus de la base roulante.

Depuis une machine cliente, il est possible de se connecter à travers le réseau en mode tuyau (TCP) sur le petit ordinateur de contrôle. Les échanges sont identiques à ceux effectués sur le port série ci-dessus.

3.2.3 Contrôle à partir de ROS

ROS permet de mettre un oeuvre une couche logiciel standardisée pour les échanges entre robots et machines de contrôle. La couche d'adaptation à ROS est à votre disposition, voir 4.1.4 page 14.

Cette couche d'adaptation tourne sur votre ordinateur principal. Les communications entre la couche d'adaptation et le robot se font à travers le réseau, comme dans le cas ci-dessus. Le cordon série du robot doit être relié au petit ordinateur de contrôle se trouvant au-dessus de la base roulante.

4 Annexes

4.1 Quelques éléments techniques à disposition

Vous vous reporterez également au site Sirien pour d'autres compléments.

4.1.1 Interface de port série

Vous pouvez télécharger aux adresses indiquées ci-dessous deux fichiers sources nommés « `serie.h` » et « serie.c » qui vous permettent d'utiliser le port série dans votre programme, que ce soit sous Linux ou sous Windows.

Adresses de téléchargement :

- http://ims.metz.supelec.fr/wiki/Depot/Gutzwiller/SIR/TL/serie.c

- http://ims.metz.supelec.fr/wiki/Depot/Gutzwiller/SIR/TL/serie.h

Il vous suffit d'intégrer des deux fichiers dans votre projet (par exemple rajouter le fichier « `serie.c` » dans le « `makefile` ») pour pouvoir utiliser les fonctionnalités. Veuillez lire dans « `serie.h` » les instructions pour utiliser le port série.

Pour utiliser le port série de la machine, vous pouvez inclure les fichiers ci-dessous dans votre projet. Vous devez définir l'une des macros _WINDOWS_ ou _LINUX_ selon la machine sur laquelle vous compilez :

- _WINDOWS_ : si vous compilez sur Windows,
- _LINUX_ : si vous compilez sur Linux.

4.1.2 Documentation des robots

Les documentations des robots se trouvent dans la salle et sont également consultable aux adresses suivantes :

- robot blanc, http://ftp.k-team.com/koala/documentation/KoalaClassicUserManual.pdf ;
- robot argenté, http://ftp.k-team.com/koala/documentation/KoalaSilverUserManual.pdf.

4.1.3 Différences entre les robots

Afin d'adapter votre programme aux deux modèles du robot, vous trouverez ici la liste des différences entre ces deux modèles. Il convient que votre programme soit paramétrable afin de proposer un comportement identique sur les deux robots.

- Unité de vitesse :
  - robot blanc : 3 mm/s ;
  - robot argenté : 4,5 mm/s.
- Unité de distance :
  - robot blanc : 0,03 mm (il faut environ 33 impulsions pour avancer d'un mm) ;
  - robot argenté : 0,045 mm (il faut environ 22 impulsions pour avancer d'un mm).
- Nombre d'impulsions par tour de roue :
  - robot blanc : 8400 ;
  - robot argenté : 5850.
- Vitesse maximale :
  - robot blanc : 155 (0,465 m/sec) ;
  - robot argenté : 128 (0,576 m/sec).
- Le bouton de sélection de mode ne se trouve pas au même endroit sur les deux robots et les deux robots n'ont pas le même nombre de modes. En revanche, dans le mode 3, les deux robots sont compatibles (hors différences notées ci-dessus).

4.1.4 Interface ROS

Une interface ROS est disponible à la fois pour commander les robots et pour accéder aux informations qu'ils peuvent fournir.

Vous pouvez télécharger cette interface ici : http://ims.metz.supelec.fr/wiki/Depot/Gutzwiller/SIR/TL/Robots/koala.tar.gz. Décompressez cette archive dans votre espace de travail ROS (`~/catkin_ws/src`).

Les explications concernant cette interface se trouvent sur la page http://malis.metz.supelec.fr/spip.php?article194.

Après avoir compilé par `( roscd; cd ..; catkin_make )`, vous pouvez lancer les services comme indiqué sur le site par :

```
roslaunch koala_node bringup.launch koala_dns:=THE_IP_OF_THE_ROBOT
```

Note : pour connaître l'adresse IP du robot, adressez-vous aux encadrants du TL.

Une fois les services lancés, vous pouvez, à la main, vérifier que le robot reconnaît bien les messages :

- Pour le faire avancer :
  `rostopic pub -1 /cmd_vel geometry_msgs/Twist -- '[0.1, 0, 0]' '[0, 0, 0]'`
- Pour le faire reculer :
  `rostopic pub -1 /cmd_vel geometry_msgs/Twist -- '[-0.1, 0, 0]' '[0, 0, 0]'`
- Pour le faire tourner vers la droite :
  `rostopic pub -1 /cmd_vel geometry_msgs/Twist -- '[0, 0, 0]' '[0, 0, -0.6]'`
- Pour le faire tourner vers la gauche :
  `rostopic pub -1 /cmd_vel geometry_msgs/Twist -- '[0, 0, 0]' '[0, 0, 0.6]'`
- Pour le stopper :
  `rostopic pub -1 /cmd_vel geometry_msgs/Twist -- '[0, 0, 0]' '[0, 0, 0]'`

4.1.5 Interface réseau

La communication réseau s'effectue par le protocole « Telnet ». Les noms des robots sur le réseau sont :

- koala1.smart.metz.supelec.fr
- koala2.smart.metz.supelec.fr
- koala3.smart.metz.supelec.fr
- koala4.smart.metz.supelec.fr

et le numéro du port est 4100.

4.1.6 Récupération des signaux pour la BCI au moyen de ROS

Rendre les enregistrements de signaux BCI disponibles

Les enregistrements se trouvent dans l'archive que vous pouvez télécharger depuis : http://ims.metz.supelec.fr/wiki/Depot/Gutzwiller/SIR/TL/Robots/acquisition_biosemi.zip. Il faut décompresser cette archive dans votre dossier de projet ROS : `~/catkin_ws/src`. Compilez ensuite l'ensemble par la commande :

```
( roscd; cd ..; catkin_make )
```

Pour faire jouer un enregistrement, tapez dans un terminal la commande :

```
( roscore & )
roscd acquisition_biosemi/
rosrun acquisition_biosemi biosemi Enregistrements/herve001.txt
```

Remarque : afin de ne pas perdre d'échantillons au début, vous devez lancer votre propre programme avant de lance le simulateur d'acquisition.

Voici les différents fichiers disponibles. Les enregistrements correspondent aux mouvements décrits ci-dessous :

herve001.txt :
— 00:00 → rien
— 00:05 → avance
— 00:10 → droite
— 00:15 → gauche
— 00:29 → rien
— 00:36 → avance
— 00:43 → gauche
— 00:52 → droite

herve002.txt :
— 00:00 → rien
— 00:07 → gauche
— 00:12 → avance
— 00:18 → droite
— 00:24 → avance
— 00:33 → droite
— 00:37 → avance
— 00:46 → gauche
— 00:54 → avance
— 01:04 → gauche
— 01:10 → avance

herve003.txt :
— 00:00 → rien
— 00:09 → droite
— 00:13 → avance
— 00:19 → gauche
— 00:24 → avance
— 00:36 → droite
— 00:41 → rien
— 00:48 → gauche
— 00:59 → avance
— 01:08 → gauche
— 01:19 → avance
— 01:31 → fin de fichier

Rendre l'acquisition BCI disponible

Vous pouvez utiliser le dispositif d'acquisition BCI avec une personne volontaire. Il faudra placer sur la tête de la personne un bonnet comportant des électrodes permettant de capter les signaux issus de son cerveau.

Le système d'acquisition fonctionne sur un ordinateur tournant sous Windows.

Il convient donc de lancer les programmes suivants :

— Sur le PC d'acquisition fonctionnant sous Windows :
Lancer : `acquisition_biosemi 1 2 --reseau --diagnostique --numerote`
Note : "1 2" indique ici d'utiliser les électrodes numérotées 1 et 2. Veuillez utiliser les numéros des deux électrodes que vous avez effectivement branchées sur le bonnet.

— Sur votre machine tourant sous Linux :
Lancer la séquence suivante :
```
( roscore & )
biosemi_talker  windows
```
Note : `windows` représente le nom de la machine d'acquisition tournant sous Windows. Veuillez le remplacer par le nom effectif de la machine que vous utilisez pour l'acquisition.

Récupérer les signaux BCI dans votre propre programme

Vous pouvez vous inspirer du programme "biosemi_listener.cpp" pour écrire votre propre programme.

Votre programme doit s'abonner au topic "biosemi_result" qui fournit des messages au format suivant :

```
uint32 numero
uint32 nombre
uint32 frequence
float64[] valeurs
string extra
```

Avec :

— **numero** : indique le numéro d'ordre du premier échantillon. Vous pouvez vérifier, au cours du déroulement de votre programme que les numéros d'ordre se suivent, ce qui signifie que vous n'avez pas perdu d'échantillon.

— **nombre** : indique le nombre d'échantillons dans le message.

— **frequence** : indique la fréquence d'échantillonnage. À noter que cette fréquence n'est pas la même selon que vous jouez des échantillons enregistrés ou que vous utilisez le système d'acquision Biosemi.

— **valeurs** : tableau contenant les échantillons. À noter qu'un échantillon comporte plusieurs valeurs numériques selon le nombre de canaux que vous aurez choisi d'utiliser. Dans les exemples pour le TL, l'acquisition se fait sur deux canaux. Un échantillon est donc composé de deux valeurs numériques, ce qui donne une taille de tableau égale à deux fois le nombre d'échantillons.

— **extra** : chaîne de caractères donnant un message en provenance du système d'acquisition. Cette chaîne est en principe vide. Si elle ne l'est pas, cela indique un problème avec le système d'acquisition. Consulter votre encadrant pour résoudre le problème.

4.1.7 Contrôler finement la console et en particulier l'arrêt du programme

En incluant le fichier `console.h` (http://www.metz.supelec.fr/metz/personnel/gutzwiller/_TUTOS_98763456_/20100506/windows/console.h) dans votre programme

principal (`#include "console.h"`), vous pouvez contrôler (sur Windows et sur Linux) l'affichage de vos message et l'arrêt de votre programme. Cela est particulièrement utile pour assurer l'arrêt de tout mouvement du robot lorsque votre programme quitte.

ATTENTION : n'utilisez pas cette possibilité si vous utilisez ROS. En effet, ROS utilise son propre système concurrent pour permettre l'arrêt du programme par la frappe de CTRL + C.

4.1.8 Afficher les images en provenance de la webcam

La commande est :
`supelec-videoserv /dev/video0 rgb 320 320 | supelec-videodisplay -`
Vous pouvez insérer un programme personnel entre l'acquisition et la restitution afin de réaliser un traitement.

Les images sont transmises les unes après les autres. Chaque image est précédée d'un en-tête de 32 octets dont le format est le suivant :
`0x10 0x00 0x00 0x00 0xFF 0xFF 0xFF 0xFF <largeur 4 octets> <hauteur 4 octets>`
`0x55 0x55 0x55 0x55 <largeur 4 octets> <hauteur 4 octets> 0xAA 0xAA 0xAA 0xAA`
L'image comporte l'ensemble des pixels (3 octets par pixel, ligne par ligne). Pour la largeur et la hauteur, le 1er octet contient les bits de poids faible, et le dernier octet les bits de poids fort. Vous pouvez consulter la page http://sirien.metz.supelec.fr/spip.php?article70.

4.1.9 Afficher les images en provenance de la caméra des robots

Les robots koala (ceux qui possèdent un CPU intégré) embarquent un serveur vidéo. Vous pouvez afficher les images vues par ces robots en utilisant la commande :

`supelec-videodisplay koala1.smart 4102`

4.1.10 Exemple de client/serveur réseau en C

Le programme que vous pouvez charger à l'adresse http://www.metz.supelec.fr/metz/personnel/gutzwiller/_TUTOS_98763456_/20101201/connect.c est un exemple de client et de serveur en C ; il fonctionne sur Windows et sur Linux. Pour la compilation, utiliser la commande :

- pour Linux : `gcc connect.c -o connect -Wall -g -D_LINUX_`
- pour Windows : `gcc connect.c -o connect -Wall -g -lwsock -D_WINDOWS_`

4.2 MFCC

4.2.1 Quelques détails sur les MFCC

Le schéma général explicitant le calcul des MFCC (Mel Filter Cepstrum Coefficients) est donné sur la figure 3. En italique, sont données aussi les tailles, parfois en deux dimensions, des signaux extraits.

La longueur des fenêtres d'analyse est typiquement de 32 ms. La longueur du pas d'avancement est de l'ordre de 16 ms.

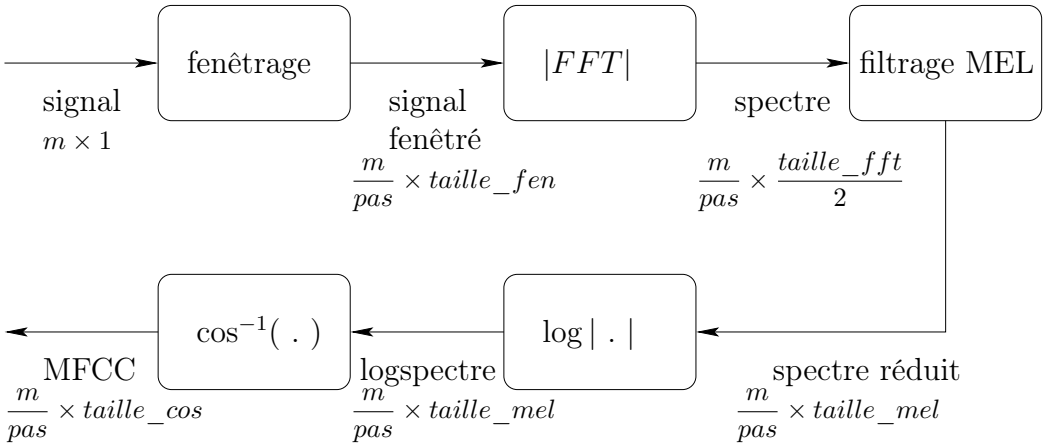


FIGURE 3 – Schéma général explicitant le calcul des MFCC

Il n'est bien sûr pas demandé de reprogrammer la FFT en C. Le package `ims-fft` de l'équipe, disponible à l'adresse http://malis.metz.supelec.fr/spip.php?article93, peut être utilisé. La iDCT peut être codée directement, car elle se calcule sur très peu de points.

4.2.2 Filtres MEL

La formule de conversion F (domaine MEL) vers f (domaine fréquentiel) est :

$$F = 2595 \log_{10}\left(\frac{f}{700} + 1\right)$$

et donc la formule inverse est :

$$f = 700\left(10^{\frac{F}{2595}} - 1\right)$$

Dans le domaine MEL, les filtres sont linéairement espacés, et de largeurs égales. Les bornes d'un filtre donné correspondent aux fréquences centrales des deux filtres qui lui sont adjacents. Dans le domaine fréquentiel, les filtres sont de forme triangulaire. Soit N_f le nombre de filtres considérés, F_{min} la plus petite fréquence MEL considérée et F_{max} la plus grande. La largeur de chaque filtre dans le domaine MEL est :

$$W = 2\frac{F_{max} - F_{min}}{N_f + 1}$$

Alors, les trois fréquences importantes, c'est-à-dire celles de chacune des deux bornes et la fréquence centrale, sont respectivement égales à, pour le filtre de numéro d'ordre i (avec $i = 1 \rightarrow N_f$) :

$$F_{centre}^{(i)} = F_{min} + i\frac{W}{2},$$

$$F_{min}^{(i)} = F_{min} + (i-1)\frac{W}{2}, \text{ et}$$

$$F_{max}^{(i)} = F_{min} + (i-1)\frac{W}{2} + W$$

et il est aisé de trouver les $f_{min}^{(i)}$, $f_{centre}^{(i)}$ et $f_{max}^{(i)}$ correspondant. Comme les filtres sont triangulaires dans le domaine fréquentiel, et de réponse unitaire pour $f_{centre}^{(i)}$, la réponse en fréquence pendant la phase ascendante a et la réponse en fréquence pendant la phase descendante b sont aisées à calculer :

$$a = \frac{f - f_{min}^{(i)}}{f_{centre}^{(i)} - f_{min}^{(i)}} \text{ pour } f \in [f_{min}^{(i)} \quad f_{centre}^{(i)}]$$

et

$$b = \frac{f_{max}^{(i)} - f}{f_{max}^{(i)} - f_{centre}^{(i)}} \text{ pour } f \in [f_{centre}^{(i)} \quad f_{max}^{(i)}]$$

4.2.3 La FFT

Il ne vous est pas demandé de programmer la FFT. Il existe des bibliothèques toutes prêtes, telle par exemple celle que vous trouvez ici :
http://malis.metz.supelec.fr/spip.php?article93

4.2.4 La DCT

$\cos^{-1}$ est une notation correspondant à la transformation en cosinus inverse d'un signal échantillonné X. Il existe un certain nombre de définitions pour la transformation en cosinus inverse. Celle qui peut être utilisée ici est :

$$x[n] = \sum_{k=0}^{N-1} w[k]X[k]\cos\left(\pi\frac{(2n-1)k}{2N}\right), \quad k = 0, \ ..., \ N-1$$

avec : $w(0) = \sqrt{\frac{1}{N}}$ et $w(k) = \sqrt{\frac{2}{N}}, \quad k = 1, ..., N-1$

4.3 DTW

Il s'agit dans un premier temps de déterminer k sons de référence. Ici, typiquement, $k = 4$, puisqu'il y a quatre ordres différents à reconnaître. Mais ceci peut-être raffiné (plusieurs sons par ordre si plusieurs locuteurs, etc.) ! Comme indiqué dans l'énoncé, vous pouvez commencer par prendre *agauche.wav*, *adroite.wav*, *enavant.wav* et *stop.wav*.

Il faut calculer une distance entre le son à reconnaître et chacun des sons de référence, et la distance la plus petite nous donne la solution.

Le problème est de définir une « distance » entre deux sons.

4.3.1 Distance locale

D'ores et déjà, nous ne nous travaillons pas directement avec les sons, mais avec les caractéristiques extraites. Supposons que le son à reconnaître ait donné lieu à l'extraction de N trames et que le son de référence k ait donné lieu à l'extraction de J^k trames. Nous formons la matrice D des distances locales $d(x_n, y_j^k)$ qui est de dimension $N \times J^k$. x_n correspond au vecteur des caractéristiques extraites pour la trame numéro n du son à reconnaître et y_j^k correspond au vecteur des caractéristiques extraites pour la trame numéro j du son de référence k. La distance considérée peut être par exemple la distance euclidienne :

$$d(x_n, y_j^k) = \sqrt{\sum_{i=1}^{I} (x_{ni} - y_{ji}^k)^2}$$

où I est le nombre de caractéristiques extraites.

D'autres distances locales peuvent être utilisées.

4.3.2 Meilleure distance globale

Pour rechercher la meilleure distance globale entre la séquence à reconnaître et la séquence de référence, il suffit alors de rechercher le chemin optimal dans la matrice D de façon à minimiser la somme des distances locales rencontrées pour aller d'un point initial (généralement $(1, 1)$, correspondant au début des deux séquences) à un point final (généralement $(N, J(k))$, correspondant à la fin des deux séquences).

Il peut être montré que la distance optimale est obtenue en calculant, pour chaque entrée (n, j) la distance cumulée $D(n, j)$ correspondant à la distance optimale qui est obtenue en comparant les deux sous-séquences correspondant aux n premiers vecteurs à reconnaître et aux j premiers vecteurs de référence. Il peut être montré que cette distance peut se calculer selon la récurrence suivante :

$$D(n, j) = d(n, j) \min_{p(n,j)} \{D(p(n, j))\}$$

où $p(n, j)$ représente l'ensemble des prédécesseurs possibles de (n, j), définis de façon à obtienr une trajectoire monotone et plausible.

Dans le cas le plus simple, et en général, nous prenons :

$$p(n, j) = \{(n - 1, j)\ (n, j - 1)\ (n - 1, j - 1)\}$$

Il y a d'autres contraintes possibles.

Sur la figure 4, nous présentons la contrainte donnée ci-dessus et un exemple de chemin obtenu avec $I = 1$ et les caractéristiques extraites [4231] et [44422231] pour les deux séquences à comparer (les valeurs des caractéristiques sont artificiellement reportées sur les axes).

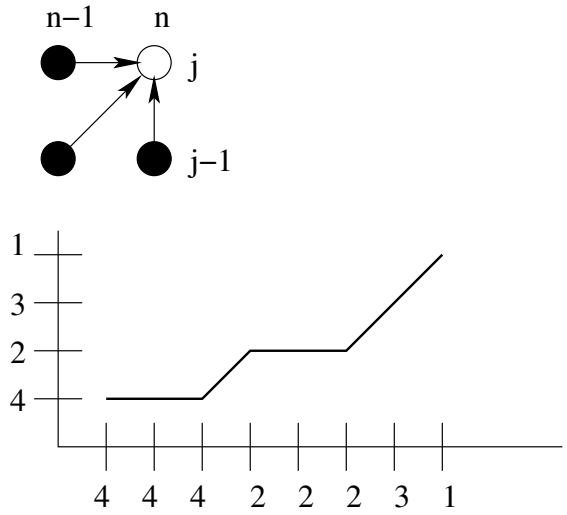


FIGURE 4 – La DTW

4.4 Prédiction linéaire

4.4.1 Introduction

L'utilisation de la prédiction linéaire constitue un moyen efficace pour caractériser un signal de parole. En effet, le modèle autorégressif qui résulte de l'estimation à court terme de cette prédiction modélise assez bien les réflexions multiples qui surviennent dans le conduit vocal à un instant donné. Celles-ci permettent d'identifier le son prononcé. Le signal de parole est alors caractérisé par une succession d'estimations à court terme des coefficients de prédiction. Ceux-ci sont relativement indépendants de la fréquence fondamentale du signal, dans les parties voisées.

4.4.2 Mise en œuvre

La prédiction linéaire peut être évaluée toutes les 10 ms dans une fenêtre temporelle de 30 ms, correspondant à la durée pendant laquelle on considère que le signal de parole est stationnaire. On utilise donc une tranche de $N = T * f_e$ échantillons pour constituer une fenêtre temporelle avec $T = 30$ ms et f_e la fréquence d'échantillonnage. Dans chaque tranche de N échantillons $x(n)$ avec $0 \leq n < N$, on se propose de trouver les K coefficients $a_K(k)$ qui minimisent l'énergie de l'erreur de prédiction $\sum_{n=1}^{N+K-1} (x(n) - \hat{x}(n))^2$ avec $\hat{x}(n) =$

$\sum_{k=1}^{K} a_K(k)x(n-k)$. On se ramène à la minimisation de la norme du vecteur $\overrightarrow{x} - X\overrightarrow{a}$ avec $\dim(\overrightarrow{x}) = (N+K-1)\times 1$, $\dim(X) = (N+K-1)\times K$, $\dim(\overrightarrow{x}) = K\times 1$ et

$$\overrightarrow{x} = \begin{pmatrix} x(1) \\ ... \\ x(N-1) \\ 0 \\ ... \\ 0 \end{pmatrix}$$

$$X = \begin{pmatrix} x(0) & 0 & ... & ... & 0 \\ x(1) & x(0) & 0 & ... & 0 \\ ... & ... & ... & ... & ... \\ x(N-1) & x(N-2) & ... & ... & ... \\ 0 & x(N-1) & x(N-2) & ... & ... \\ ... & ... & ... & ... & ... \\ 0 & 0 & ... & 0 & x(N-1) \end{pmatrix}$$

$$\overrightarrow{a} = \begin{pmatrix} a_K(1) \\ ... \\ a_K(K) \end{pmatrix}$$

La solution est donnée par les moindres carrés où le vecteur $\overrightarrow{a}$ est solution du système linéaire $X^tX\overrightarrow{a} = X^t\overrightarrow{x}$. La structure particulière de la matrice (de Toëplitz) permet une résolution plus rapide avec l'algorithme de Levinson-Durbin.

4.4.3 Algorithme de Levinson-Durbin

Dans le système à résoudre, la matrice X^tX peut s'exprimer sous la forme

$$X^tX = N\begin{pmatrix} \hat{\phi}_x(0) & \hat{\phi}_x(1) & ... & ... & \hat{\phi}_x(K-1) \\ \hat{\phi}_x(1) & \hat{\phi}_x(0) & ... & ... & \hat{\phi}_x(K-2) \\ ... & ... & ... & ... & ... \\ \hat{\phi}_x(K-1) & \hat{\phi}_x(K-2) & ... & ... & \hat{\phi}_x(0) \end{pmatrix}$$

et le vecteur $X^t\overrightarrow{x}$ sous la forme

$$X^t\overrightarrow{x} = N\begin{pmatrix} \hat{\phi}_x(1) \\ ... \\ \hat{\phi}_x(K) \end{pmatrix}$$

avec $\hat{\phi}_x(k) = \frac{1}{N}\sum_{n=0}^{N-k-1} x(n)x(n+k)$. Ainsi, $\hat{\phi}_x(k)$ est un estimateur de la fonction d'autocorrélation de x : $\phi_x(k) = \mathrm{E}\left\{x(n)x(n+k)\right\}$.

L'algorithme de Levinson-Durbin permet d'obtenir les K coefficients de prédiction en résolvant successivement le problème de la prédiction à n coefficients pour $1 \leq n \leq K$. Notons $a_n(p)$ avec $1 \leq p \leq n$ les coefficients de prédiction obtenus lorsqu'on cherche une

prédiction à l'ordre n. Ceux-ci peuvent être calculés à partir des coefficients associés à une prédiction d'ordre $n-1$. L'algorithme donne aussi les coefficients de réflexion k_n appelés aussi coefficients de corrélation partielle (PARCOR). Ceux-ci vérifient la condition $|k_n| < 1$. Ils peuvent être utilisés aussi pour caractériser le signal. L'algorithme est le suivant :

Initialisation :

$\alpha_1 = \hat{\phi}_x(0)$

$k_1 = -\dfrac{\hat{\phi}_x(1)}{\hat{\phi}_x(0)}$

$a_1(1) = -k_1$

Récursion :

Pour n variant de 2 à K

$\alpha_n = \alpha_{n-1}(1 - k_{n-1}^2)$

$k_n = -\dfrac{1}{\alpha_n}\left(\hat{\phi}_x(n) - \sum_{p=1}^{n-1} a_{n-1}(p)\hat{\phi}_x(n-p)\right)$

$a_n(n) = -k_n$

Pour p variant de 1 à $n-1$

$a_n(p) = a_{n-1}(p) + k_n a_{n-1}(n-p)$

fin Pour

fin Pour

4.4.4 Lien entre prédiction et analyse spectrale

Une estimation $\hat{\sigma}_e^2$ de la variance de l'erreur de prédiction peut être obtenue à partir des coefficients de prédiction :

$$\sigma_e^2 = \mathrm{E}\left\{(x(n) - \hat{x}(n))^2\right\} = \mathrm{E}\left\{x(n)(x(n) - \hat{x}(n))\right\}$$

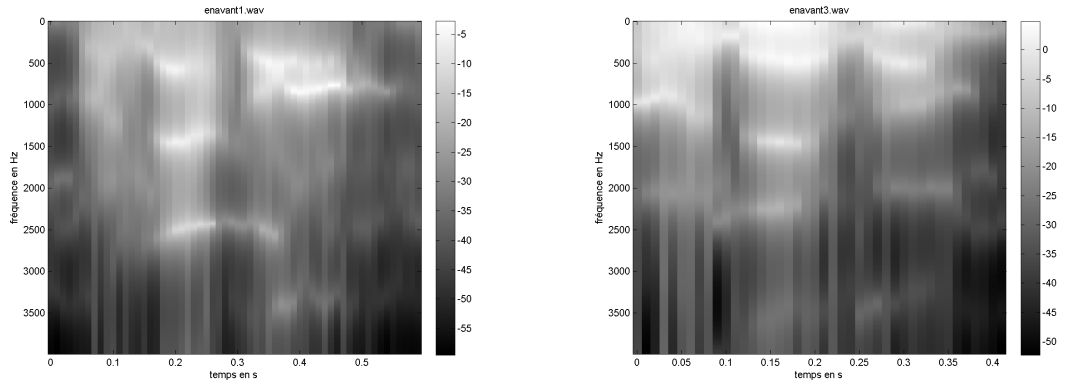
soit

$$\hat{\sigma}_e^2 = \hat{\phi}_x(0) - \sum_{k=1}^{K} a_K(k)\hat{\phi}_x(k)$$

En faisant l'hypothèse que le signal d'erreur de prédiction est un bruit blanc, on obtient alors une estimation paramétrique de la densité spectrale de puissance du signal :

$$\hat{\Phi}_x(\nu) = \frac{\hat{\sigma}_e^2}{\left|1 - \sum_{k=1}^{K} a_K(k)\exp(2\pi j\nu k)\right|^2}$$

Avec une fréquence d'échantillonnage $f_e = 8\text{kHz}$ et un ordre $K = 10$, on fait apparaître plutôt les propriétés du conduit vocal (formants ou résonances) indépendamment par exemple de la fréquence fondamentale dans les parties voisées. On le constate en observant les propriétés spectrales associées à la prononciation de la même phrase par deux locuteurs (voir figure 5). Les coefficients de prédiction sont donc bien caractéristiques des propriétés spectrales à court terme du signal de parole.



(a) (b)

FIGURE 5 – Exemples de spectres

4.5 Algorithme des K-moyennes

Processus de minimisation Il s'agit d'un algorithme itératif, qui utilise à chaque itération l'ensemble des données ξ_i disponibles (algorithme dit *batch* ou hors ligne). Le cardinal K de W est prédéfini.

Initialisation $m = 0$;
$\widehat{\mathbf{W}}_m = [\xi_{\sigma(1)}, \ldots, \xi_{\sigma(K)}]$, où σ est une permutation de $[\![1, N]\!]$.
$\widehat{\Psi}_1 = \Psi(\widehat{\mathbf{W}}_m)$;

Etape 1 Etant donné l'ensemble de prototypes $\{\widehat{\mathbf{w}}_j^m\}_{1 \leq j \leq K}$, générer l'ensemble de prototypes $\{\widehat{\mathbf{w}}_j^{m+1}\}_{1 \leq j \leq K}$ de la manière suivante :

(a) trouver la partition optimale C des ξ_i telle que le *cluster* C_k soit l'ensemble de tous les ξ_i appartenant au polyhèdre de Voronoï $V_k^{(p)}(\widehat{W}_m)$ de $\widehat{\mathbf{w}}_k$,

(b) calculer les nouveaux prototypes $\{\widehat{\mathbf{w}}_j^{m+1}\}_{1 \leq j \leq K}$:

$$\widehat{\mathbf{w}}_k^{m+1} = \frac{1}{N_k} \sum_{\xi_i \in C_k} \xi_i \tag{3}$$

avec

$$N_k = \text{card}(C_k). \tag{4}$$

(c) $\widehat{\Psi}_0 = \widehat{\Psi}_1$;
$m = m + 1$;
$\widehat{\mathbf{W}}_m = [\widehat{\mathbf{w}}_1^m, \ldots, \widehat{\mathbf{w}}_K^m]$;
$\widehat{\Psi}_1 = \Psi(\widehat{W}_m)$.

Etape 2 Si $\widehat{\Psi}_0 - \widehat{\Psi}_1 > \epsilon \geq 0$, aller à l'étape 1, sinon retourner $\widehat{\Psi}_1$ et $\widehat{\mathbf{W}}_m$.

Remarquons que l'étape 1(a) correspond à la condition du plus proche voisin et l'étape 1(b) à la règle du centroïde, qui sont des conditions nécessaires pour qu'un quantificateur soit optimal.

Une étude de la convergence de cet algorithme est présentée dans [4], où il est en particulier démontré que ce processus, applique un algorithme d'optimisation de Newton avec la règle d'adaptation (3).

Remarquons qu'on ne peut pas définir de bornes analogues à celles utilisées en théorie de l'apprentissage statistique [5] pour la généralisation [6].

4.6 Algorithme des K plus proches voisins

Etant donné un vecteur ξ composé de m attributs, la méthode consiste à déterminer, parmi les K plus proches points de ξ, au sens d'une distance d sur R^m, la classe la plus représentée. Si $K = 1$, le point ξ est affecté à la classe de son plus proche voisin. L'algorithme s'énonce ainsi :

Données : Un dictionnaire contenant les vecteurs de référence

1. Calculer la distance entre le vecteur ξ et tous les vecteurs de référence.
2. Garder les K plus petites distances
3. Attribuer au vecteur ξ la classe la plus représentée parmi les K vecteurs sélectionnés.

La distance entre deux vecteurs X=$\{x_1, x_2, ..., x_m\}$ et Y=$\{y_1, y_2, ..., y_m\}$ est définie de la manière suivante :

$$d_n(X, Y) = (\sum_{i=1}^{m} |x_i - y_i|^n)^{1/n}$$

Pour n = 1, on obtient la distance de Hamming :

$$d_1(X, Y) = \sum_{i=1}^{m} |x_i - y_i|$$

Pour n = 2, on obtient la distance Euclidienne :

$$d_2(X, Y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

4.7 Algorithme du perceptron multicouches

4.7.1 Présentation

Le perceptron multicouches (PMC) est composé d'une couche d'entrée, d'une couche de sortie et d'un nombre variable de couches cachées (figure 6). Le nombre de neurones des couches d'entrée et de sortie est fixé par le problème à traiter. Celui des couches

cachées est déterminé expérimentalement. La connectivité entre couches peut être totale, partielle et aléatoire, partielle et structurée ou partielle structurée à poids partagés. Les neurones sont modélisés à l'aide d'automates continus. Les entrées et les sorties prennent leur valeur dans l'intervalle [-1, 1]. Les poids w_{ij} sont initialisés aléatoirement.

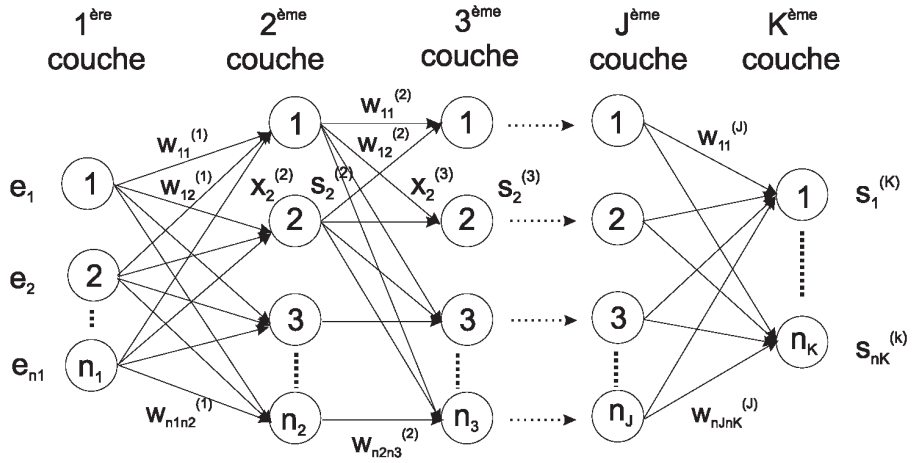


FIGURE 6 – Structure du perceptron multi-couches

Un neurone appartenant à la deuxième couche calcule sa sortie selon les expressions suivantes :

$$s_k^{(2)} = f(x_k^{(2)}) \tag{5}$$

avec :

$$f(x_k^{(2)}) = \frac{1}{1 + e^{-x_k^{(2)}}} \tag{6}$$

et :

$$x_k^{(2)} = \sum_{j=1}^{n_1} w_{jk}^{(1)} e_j \tag{7}$$

Un neurone appartenant à la couche l avec $l = (2, 3, ..., J, K)$ calcule sa sortie de la manière suivante :

$$s_k^{(l)} = f(x_k^{(l)}) \tag{8}$$

avec :

$$f(x_k^{(l)}) = \frac{1}{1 + e^{-x_k^{(l)}}} \tag{9}$$

et :

$$x_k^{(l)} = \sum_{j=1}^{n_{l-1}} w_{jk}^{(l-1)} s_j^{(l-1)} \tag{10}$$

La règle d'apprentissage du perceptron multicouches est une règle d'apprentissage supervisée, qui se fait par correction d'erreur. Le but de l'algorithme d'apprentissage est de faire coïncider la sortie réelle (s) avec la sortie désirée (d), et ce pour chaque vecteur du corpus d'apprentissage. Pour l'ensemble du corpus d'apprentissage on minimise l'erreur :

$$E(\mathbf{w}) = \sum_{p=1}^{P_A} E^p(\mathbf{w})$$

où P_A représente le nombre d'exemples de la base d'apprentissage et $E^p(\mathbf{w})$ l'erreur quadratique sur un exemple :

$$E^p(\mathbf{w}) = \sum_{l=1}^{n_K} (s_l^p - d_l^p)^2$$

n_K étant le nombre de neurones de la couche de sortie.

On résume ci-dessous les différentes étapes de l'algorithme d'apprentissage du perceptron multicouches :

1. Initialiser les poids du réseau aléatoirement. Typiquement les poids sont choisis dans un petit intervalle centré autour de 0 et uniformément distribué dans cet intervalle.
2. Pour tous les exemples p de l'ensemble d'apprentissage
   (a) Présenter un couple de vecteur (e^p,d^p) au réseau. Le vecteur (e^p) étant le vecteur d'entrée et le vecteur (d^p) le vecteur de sortie désirée.
   (b) Calculer l'état du réseau (s^p) par propagation directe. Le vecteur (e^p) est propagé de l'entrée du réseau vers la sortie selon les expressions 5, 6, 7 et 8, 9, 10.
   (c) Calculer l'erreur de sortie du réseau

   $$E^p(\mathbf{w}) = \sum_{l=1}^{n_K} (s_l - d_l)^2$$

   où n_K représente le nombre de neurones de la couche de sortie.
   (d) Calculer le gradient partiel de la fonction de coût :

   $$\nabla_E(\mathbf{w}) = \left(\frac{\partial E^p(\mathbf{w})}{\partial w_{11}^{(1)}}, \frac{\partial E^p(\mathbf{w})}{\partial w_{12}^{(1)}}, ..., \frac{\partial E^p(\mathbf{w})}{\partial w_{n_J n_K}^{(J)}} \right)^T$$

   (e) Modifier les poids du réseau afin de minimiser la fonction de coût :

   $$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \nabla_E(\mathbf{w}(t))$$

   où η représente le pas d'apprentissage.
3. Retourner en (2) jusqu'à convergence de l'algorithme.

4.7.2 Utilisation de la Toolbox Neural Network de Matlab

L'exemple ci-dessous montre comment utiliser les fonctions de la *Toolbox Neural Network* de *Matlab* pour simuler le fonctionnement d'un perceptron multicouches. Le réseau de neurones est composé d'une couche d'entrée de 19 neurones, d'une couche cachée de 10 neurones et d'une couche de sortie de 4 neurones. L'apprentissage du réseau de neurones se fait avec 3000 vecteurs de 19 composantes chacun et le test du réseau de neurones se fait avec 2000 vecteurs.

La fonction `newff` permet de créer le perceptron multicouches :

```
net = newff(entree, sortie, 10, {'tansig' 'tansig'}, 'trainscg');
```

Les paramètres d'appel de la fonction sont :

- `entree` est une matrice de 19 lignes et 5000 colonnes qui contient les vecteurs d'entrée,
- `sortie` est une matrice de 4 lignes et 5000 colonnes qui contient les vecteurs de sortie,
- `10` correspond au nombre de neurones de la couche cachée,
- `{'tansig' 'tansig'}` correspond au type de fonction d'activation utilisé pour modéliser les neurones de la couche cachée et de la couche de sortie (`tansig` correspond à la fonction tangente hyperbolique).
- `'trainscg'` permet de préciser le type d'algorithme utilisé pour calculer les poids du réseau (`traincsg` correspond à l'algorithme du gradient conjugué).

La fonction retourne une objet de type `network` qui est stocké dans la variable `net`. Certains des champs de la variable `net` sont initialisés avec des valeurs par défaut. On peut modifier la valeur de ces champs de la manière suivante :

```
net.trainParam.epochs = 1000;     % Le nombre de cycle d'apprentissage est fixé à
                                  % 1000

net.trainParam.lr = 0.02;         % Le pas d'apprentissage est égal à 0.02

net.trainParam.show = 100;        % Des informations sur les performances du réseau
                                  % sont affichées tous les 100 cycles d'apprentissage

net.trainParam.goal = 1e-10;      % L'algorithme d'apprentissage s'arrête lorsque
                                  % l'erreur quadratique moyenne est inférieure à
                                  % 1e-10

net.trainParam.min_grad = 1e-10; % L'algorithme d'apprentissage s'arrête lorsque le
                                  % module du gradient est inférieur à 1e-10

net.divideParam.trainRatio = 3000 / 5000; % On utilise 3000 exemples de la matrice
net.divideParam.valRatio = 0;             % 'entree' pour faire l'apprentissage
net.divideParam.testRatio = 2000 / 5000;  % du réseau de neurones et les 2000
                                          % exemples restant pour le test.
```

```
                                    % On n'utilise pas d'ensemble de
                                    % validation.
```

La fonction `train` permet de faire l'apprentissage du réseau de neurones :

```
net = train(net, entree, sortie);
```

La fonction `sim` permet de tester le réseau de neurones :

```
entree_test(1: 2000) = entree(3001: 5000);
sortie_test(1: 2000) = sortie(3001: 5000);
[rt, pf, af, errort, perft] = sim(net, entree_test, [], [], sortie_test);
```

Les paramètres d'appel de la fonction sont :

- `entree_test` est une matrice de 19 lignes et 2000 colonnes qui contient les vecteurs d'entrée de l'ensemble de test,
- `sortie_test` est une matrice de 4 lignes et 2000 colonnes qui contient les vecteurs de sortie de l'ensemble de test,

La fonction renvoie les variables suivantes :

- `rt` est une matrice de 4 lignes et 2000 colonnes qui contient les sorties calculées par le réseau de neurones,
- `errort` est une matrice de 4 lignes et 2000 colonnes qui contient l'écart entre les sorties désirées et les sorties calculées par le réseau de neurones,
- `perft` est l'erreur quadratique moyenne calculée sur l'ensemble de test.

4.8 Synthèse d'un filtre passe-bande

Pour synthétiser un filtre passe-bande, on se donne sa bande passante Δf, sa fréquence centrale f_c à laquelle le gain vaut 1, la fréquence d'échantillonnage f_e et le gain maximum G à l'extérieur de la bande passante. On recherche une fonction de transfert de la forme

$$H(z) = \frac{b_0}{1 + a_1 z^{-1} + a_2 z^{-2}} = \frac{(1 - r^2)\sin\theta}{1 - 2r\cos\theta\, z^{-1} + r^2 z^{-2}} \tag{11}$$

pour réaliser le filtre passe-bande. On obtient alors les équations de synthèse (approximation valable pour des faibles bandes passantes) :

$$r^2 = 1 - \frac{2\pi G \Delta f}{f_e} \tag{12}$$

$$\theta = 2\pi \frac{f_c}{f_e} \tag{13}$$

soit

$$b_0 = \frac{2\pi G \Delta f}{f_e} \sin\left(\frac{2\pi f_c}{f_e}\right) \tag{14}$$

$$a_1 = -2\sqrt{1 - \frac{2\pi G \Delta f}{f_e}} \cos\left(\frac{2\pi f_c}{f_e}\right) \tag{15}$$

$$a_2 = 1 - \frac{2\pi G \Delta f}{f_e} \tag{16}$$

L'équation de fonctionnement du filtre avec une entrée $x(n)$ et une sortie $y(n)$ est alors :

$$y(n) = b_0 x(n) - a_1 y(n-1) - a_2 y(n-2). \tag{17}$$

Pour un signal de la forme

$$x(n) = A \sin\left(2\pi \frac{f_c}{f_e} n + \varphi\right) + \sigma b(n) \tag{18}$$

avec φ une variable aléatoire uniforme sur $[0, 2\pi]$ et $b(n)$ un bruit gaussien de variance unité décorrélé de φ, le signal d'entrée a une variance égale à

$$\frac{A^2}{2} + \sigma^2 \tag{19}$$

et le signal de sortie a une variance égale à

$$\frac{A^2}{2} + \sigma^2 \int_{-\frac{1}{2}}^{\frac{1}{2}} |H(e^{2\pi j\nu})|^2 \, d\nu \tag{20}$$

avec

$$\int_{-\frac{1}{2}}^{\frac{1}{2}} |H(e^{2\pi j\nu})|^2 \, d\nu = \frac{(1-r^4)(\sin\theta)^2}{(1+r^2+2r\cos\theta)(1+r^2-2r\cos\theta)} \tag{21}$$

et

$$H(z) = \frac{(1-r^2)\sin\theta}{1 - 2r\cos\theta \, z^{-1} + r^2 z^{-2}}. \tag{22}$$

Avec les approximations proposées, on obtient

$$\int_{-\frac{1}{2}}^{\frac{1}{2}} |H(e^{2\pi j\nu})|^2 \, d\nu \cong \frac{\pi G \Delta f}{f_e}. \tag{23}$$

Pour un signal de la forme $x(n) = A \sin\left(2\pi \frac{f_c}{f_e} n\right)$ pour $n \geq 0$, le signal de sortie a une enveloppe de la forme $A(1 - r^n) = A(1 - e^{-n/\tau})$ avec une constante de temps réduite

$$\tau \cong \frac{f_e}{\pi G \Delta f}. \tag{24}$$

Il y a donc un compromis entre l'atténuation apportée au bruit et le temps de réponse sachant que :

$$\tau \int_{-\frac{1}{2}}^{\frac{1}{2}} |H(e^{2\pi j\nu})|^2 \, d\nu \cong 1. \tag{25}$$

Références

[1] René Boite, Hervé Bourlard, Thierry Dutoit, Joël Hancq, and Henri Leich, *Traitement de la parole*, Presses polytechniques et universitaires romandes, 2000.

[2] A. Gersho and R.M. Gray, *Vector quantization and signal compression*, Dordrecht, Netherlands, 1992.

[3] D. Zhu, J. Bieger, G. Garcia Molina, and R.M. Aarts, "A survey of stimulationmethods used in ssvep-based bcis," *Computational Intelligence and Neuroscience*, vol. 2010, 2010, 12 pages.

[4] L. Bottou and Y. Bengio, "Convergence properties of the k-means algorithms," in *Proceedings of the 9th Conference on Neural Information Processing Systems (NIPS)*, Denver, CO, USA, 1995, pp. 585–92.

[5] V. Vapnik, *Statistical Learning Theory*, Wiley-Interscience, 1998.

[6] U. Von Luxburg and S. Ben-David, "Towards a statistical theory of clustering," in *Proceedings of the PASCAL Workshop on Statistics and Optimization of Clustering*, London, UK, 2005, 6 pages.