UNIVERSITÉ LAVAL

Département de génie électrique et de génie informatique
Faculté des sciences et de génie
Université Laval
1065, avenue de la Médecine
Québec (Québec) G1V 0A6
Téléphone : +1 (418) 656-2984
Télécopieur : +1 (418) 656-3159
Courrier électronique : gel@gel.ulaval.ca

Vocal Activity Algorithm
Module commande de la parole

Development and integration of a voice command for users with speech disabilities (dystrophy, paralysis, etc.) on the JACO robotic arm

Sous la tutelle de Benoit Gosselin

UNIVERSITÉ LAVAL

Département de génie électrique et de génie informatiques
Faculté des sciences et de génie
Université Laval

Table des matières

UNIVERSITÉ LAVAL

Département de génie électrique et de génie informatiques
Faculté des sciences et de génie
Université Laval

Résumé

L'objectif de ce rapport est de confronter les performances de mon algorithme de Détection d'Activité de la Voix dans une trame audio (DAV ou Vocal Activity Detection en anglais). Le DAV est un algorithme qui vise à extraite d'une trame audio tous les tronçons où il y a présence de voix et cherche à accélérer le processus de reconnaissance de la parole. En effet, en ne se concentrant que sur les tronçons de voix, on ne cherche pas à reconnaitre des silences entre des mots, ... mais que des mots, phonèmes, ... (cela dépend de la manière dont est réalisé l'algorithme de reconnaissance de la parole).

Actuellement, plusieurs méthodes sont utilisées pour réaliser cette opération de détection de flux de parole dans différents domaines (temporels et fréquentiels). Le domaine utilisé va bien sûr impacter sur la rapidité de l'algorithme. Le domaine temporel ne nécessitant pas de calcul complexe, il rendra notre algorithme plus rapide néanmoins il sera plus sensible au bruit. Le domaine fréquentiel est quant à lui moins sensible au bruit mais est plus gourmand en calcul. Les méthodes les plus connues dans le domaine temporel sont l'énergie à court terme combinée aux taux de passage à zéro (Short time Energy and Zero Crossing Rate) et dans le domaine fréquentiel : l'Entropie (Spectral Entropy), le Spectral Centroïd (qui est une sorte d'indicateur du "centre de masse" du spectre) mais aussi les Mel Feature Cepstrum Coefficents (MFCC).

Mon partis pris a été de choisir de travailler dans le domaine fréquentiel essentiellement afin d'avoir un algorithme très robuste au bruit et de combiné l'Entropy au MFCC. De plus, même si mes deux méthodes sont très peu sensible au bruit, elles perdent un peu de leur immunité à partir d'un certain SNR (Signal Noise Ratio ou Rapport de Signal à Bruit en français). De fait, afin de pouvoir travailler dans des environnements assujettis à des SNR très faible, j'ai donc généré une fonction d'initialisation qui récupère l'empreinte du bruit environnant et s'en sers de vecteurs modèles. Les vecteurs extraits des deux méthodes sont donc comparés à ces modèles de bruit par distance euclidienne ou par corrélation pour chaque trame. Les vecteurs modèles sont mis à jour à chaque trame pour suivre l'évolution des éléments en entrées.

L'étude comparative de mon modèle se fera par confrontation à quatre modèles et avec des contraintes de bruits.

1 Introduction

Ce rapport s'organise de la façon suivante. Tout d'abord quelques notions théoriques nécessaires à la bonne compréhension du sujet seront introduites. Un état de l'art non exhaustif de la détection de l'activité vocale sera présenté. Ensuite, les différents travaux réalisés seront décrits. Ces travaux reposent essentiellement sur les Mel Feature Cepstrum Coefficient (MFCC) et l'Entropie, dans l'objectif de rejeter toutes les trames audio non utiles. Les performances du VAD des méthodes proposées seront exposées puis comparées à des méthodes de références. Ce sera alors l'occasion de soulever des difficultés liées aux différentes méthodes mais aussi des améliorations qui pourraient y être éventuellement apportés.

2 Notions théoriques

2.1 Traitement à court-terme du signal vocal

On peut assez facilement constater que la forme d'onde d'un signal vocal met en évidence son caractère non stationnaire. Étant donné son caractère non stationnaire, un étude à long terme ou globale est généralement inefficaces. L'hypothèse la plus utilisée dans le traitement de la parole est le fait que les propriétés du signal vocal change lentement dans le temps [4]. Cette hypothèse conduit à un traitement à court terme. De plus, l'analyse à court terme permet de respecter le caractère temps réel que nous souhaitons donner à notre algorithme par la suite.

2.2 Méthodes dans le domaine temporel

Les méthodes ci-dessous sont une liste non exhaustive des méthodes temporelles utilisées. Elles sont néanmoins celle le plus couramment utilisées. Les définitions de chaque méthode est extraite de l'article suivant [7]

2.2.1 Énergie à court terme

L'énergie court-terme est un paramètre qui reflète les variations d'amplitude dans le signal vocal. Elle fut un de premiers paramètres utilisés dans la détection d'activité vocale. Elle permet de classifier les trames voisées et non voisées. Elle est définit de la manière suivante :

$$E_n = \sum_{k=n-N+1}^{n} [x(k).w(n-k)]^2 \tag{1}$$

où $w(n-k)$ est la fenêtre d'analyse, n est l'échantillon sur lequel est centré la fenêtre d'analyse et N la taille de la fenêtre d'analyse.

2.2.2 Taux de passage par zéro (Zero Crossing Rate, ZRC)

Le ZRC compte le nombre de passage par zéro du signal. Les trames de voisées ont un taux de passage à zéro faible alors l'inverse des trames non voisées qui ont un ZRC élevé. On peut définir le ZRC de la manière suivante :

$$Z_n = \sum_{m=-\infty}^{infty} |sgn[x(m)] - sgn[x(m-1)]lw(n-m) \tag{2}$$

où

$$sgn[x(n)] = \begin{cases} 1 & x(n) \geq 0 \\ -1 & x(n) < 0 \end{cases} \tag{3}$$

et

$$w(n) = \begin{cases} 1/(2N) & 0 \leq n \leq N-1 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

2.3 Domaine fréquentiel

2.3.1 Entropie Spectrale

Nous ne pouvons pas parler d'Entropie spectrale sans parler auparavant de l'entropie introduite par Claude Shannon dans sa théorie de l'information. Cette dernière permet de mesurer la quantité d'information contenue dans un signal aléatoire. Elle est définit de la manière suivante :

$$H(x) = \sum_k p(x_k).log_2[P(x_k)] \tag{5}$$

où $x = x_{k0 \leq k \leq N-1}$ est une série temporelle, fréquentielle ou autre et où $P(k)$ est la probabilité d'un certain état x_k.

Travaux de Shen J, Hung J et Lee J [9] Shen J., Hung J. et Lee J. ont été les premiers a utilisé l'entropie dans le cadre de la détection de la parole. Ils ont démontré que l'entropie d'un flux de parole diffère de celui d'un flux sans parole. En effet, leur étude a montré que la structure de la voix reflète d'une organisation que l'on ne retrouve pas dans la structure spectrale du silence.

L'entropie spectrale L'entropie de Shannon mesurant la quantité d'information contenue dans un signal aléatoire, il est plus d'usage d'utiliser l'entropie spectrale : la structure harmonique d'un segment de voix n'apparaissant que dans son spectrogramme. L'entropie spectrale est définit tout d'abord par la transformée de Fourier à court terme (Short time Fourier Transform) :

$$S(k,l) = \sum_{n=1}^{N} h(n)s(n-l) \exp \frac{-j2\pi kn}{N} \text{ avec } 0 \leq k \leq K-1 \text{et } K = N \tag{6}$$

$S(k,l)$ représente l'amplitude de la k^{ime} composante fréquentielle, pour la l^{ime} trame d'analyse. $s(n)$ est l'amplitude du signal au temps n. N, le nombre de points considérés pour la transformée de Fourier. $h(n)$ est la fenêtre d'analyse (généralement, une fenêtre de hamming).

Energie spectrale On définit l'énergie spectrale de la manière suivante :

$$S_{energy}(k,l) = |S(k,l)|^2 \text{ avec } 1 \leq k \leq \frac{N}{2} \tag{7}$$

P(k,l) La probabilité associée à chaque composante spectrale est obtenue en normalisant :

$$P(k,l) = \frac{S_{energy}(k,l)}{\sum_{i=1}^{\frac{N}{2}} S_{energy}(i,l)} \text{ avec } 1 \leq i \leq \frac{N}{2} \text{ et} \sum_i P(i,l) = 1 \text{ pour tout } l. \tag{8}$$

Entropie Spectrale L'entropie spectrale s'appuie sur l'entropie de Shannon :

$$H(l) = \sum_{i=1}^{\frac{N}{2}} P(i,l). \log_2 |P(i,|)| \tag{9}$$

Il est plus souvent d'usage de considérer l'opposé de l'entropie pour obtenir des profils analogues à ceux de l'intensité.

Limitation Une des limitations de l'entropie spectrale exposé par Ouzounov [3] est que l'entropie d'une trame sans parole bruitée par un bruit blanc coloré peut-être équivalente à celle d'une trame avec parole mais bruitée.

2.3.2 Spectral Centroïd

Le "centroïde spectral" C_i de la i^{ime} trame est défini comme le centre de gravité de son spectre.

$$C_i = \frac{\sum_{k=1}^{N}(k+1)X_i(k)}{\sum_{k=1}^{N}X_i(k)}. \tag{10}$$

$X_i(k)$ avec $k = 1, ..., N$ sont les coefficients de la i^{ime} courte trame de la transformée de Fourier discrète (DFT). N est la longueur de la trame. Le centroïde spectral est une mesure de la position spectrale où des valeurs élevées correspondent à des sons plus brillant.

Ces explications sont issues de l'article de Theodoros Giannakopoulos [6]. Son algorithme et son étude seront utilisés par la suite pour comparer les résultats de notre algorithme.

2.3.3 Mel Feature Cepstral Coefficients (MFCC)

Les explications qui suivent sont issues et pour la plupart recopier du site *practicalcryptography.com* [1].

Les Mel Features Cepstral Coefficients (MFCCs) sont des caractéristiques majoritairement utilisées en reconnaissance de parole automatique. Les MFCCS ont été introduites par Davis and Mermelstein dans les années 80, et ont été "l'état de l'art" depuis. Avant la mise en place des MFCC, Les coefficients de prédiction linéaire (LPC) et les coefficients cepstaux de prédilection linéaire (LPCC) constituaient la caractéristique principale de reconnaissance automatique de la parole (ASR), en particulier avec les classificateurs HMM .

Etape pour déterminer les MFCCs Pour déterminer les MFCCs, ll faut commencer, comme tout au long de notre étude, par sectionner le signal en courte trame se recouvrant les unes aux autres (overlapping). Pour chaque trame, on calcule l'estimation du périodogramme du spectre de puissance (en d'autres termes, on regarde la distribution de notre périodogramme). Ensuite, on applique un banc de filtres Mel aux spectres de puissances, somme de l'énergie dans chaque filtre. On prend le logarithme de toutes les énergies du banc de filtre. Puis la transformée en cosinus inverse (DCT). On ne garde que les 12 premiers coefficients en omettant le premier.

3 Travaux réalisés

Cette partie va aborder du travail réalisé sur l'algorithme de détection vocale (VAD). Une grande partie est inspirée de méthodes et modèles existants. En ce qui concerne le modèle de bruit de fond avec mise à jour à chaque trame, il est extrait des travaux Hongzhi Wang et Yuchao Xu, Meijing Li [11]. J'ai donc transposé leur travaux à l'entropie afin de la rendre plus robuste au bruit. Nous avions parler d'une des limitations de l'entropie un peu plus que nous avons cherché à combler par cette méthode.

3.1 Fonctionnement général de l'algorithme

Avant de rentrer des explications plus approfondis sur les méthodes utilisées ainsi que leurs outils, nous allons expliquer globalement le fonctionnement de notre algorithme.

Tout d'abord, une première fonction *recorder.m* [2] permet l'acquisition de donnée par la carte son de mon laptop et de sauvegarder des données dans des veccteurs. Deux acquisition sont réalisées :

- Acquisition du bruit environnant
- Acquisition d'une trame de voix durant 3s (le temps est ici arbitraire. D'autant plus, que par la suite nous chercherons à être en temps réel).

Une fois ces deux acquisitions opérées, nous utilisons celle de bruit de fond afin de générer un vecteur $MFCCs_{noise}$. Pour cela, nous "découpons" le signal acquis en plusieurs trames de 15ms avec un recouvrement de 10ms afin de respecter l'hypothèse du caractère stationnaire exprimé un peu plus haut. Ces trames (ou fenêtres) sont ensuite envoyées à un algorithme : $short_time_Fourier_transform.m$ pour réaliser une transformée de Fourier (l'appellation de la fonction est un peu faussée car nous sommes déjà dans un cas de court terme). Enfin, on extrait les Mel Feature Cepstral Coefficients du signal de bruit de fond (toutes les trames) que nous moyennons. Ce vecteur est donc la moyenne des coefficients cepstraux du bruit (en considérant que seule du bruit de fond est présent dans cette acquisition). On opère similairement pour l'entropie, en calculant la valeur moyenne de l'entropie pour le signal de bruit de fond (on réalise la moyenne sur l'ensemble des trames).

Pour l'acquisition d'une trame de voix, on réalise aussi le "découpage" en trame avec chevauchement. Pour chaque trame, on calcule l'entropie et les coefficients cepstaux. On compare le vecteur de coefficients cepstraux et la valeur de l'entropie de chaque trame à respectivement, le vecteur de bruit de fond de la moyenne des coefficients cepstraux et à la valeur moyenne du bruit de fond de l'entropie. On utilise pour cela, respectivement, la corrélation et la distance euclidienne.

On applique ensuite des seuils pour chacune des deux variables évoluant au cours du temps afin d'être le plus pertinent possible. Chaque trame qui est au dessus du seuil est labellisée d'un "un" sinon elle l'est d'un "zéro". Le vecteur contenant les labels est ensuite envoyé à une fonction permettant de corriger les valeurs aberrantes (par exemple un "un" perdu au milieu de "zéro" et vis-versa. L'algorithme corrige jusqu'à plusieurs paquets de valeurs aberrantes en fonction des paramètres qu'on lui rentre en argument). Pour cela, on s'appuie du poids des labels au nombres paires entourant chaque label considéré.

Une fois l'opération de correction de label réalisée, nous n'avons plus qu'à extraire les segments (en ms et non en trame comme nous avions précédemment) en considérant nos labels de "un" comme de la

voix et les autres comme du silence, du bruit et des parasites sur le signal. Enfin, un dernier algorithme tri les segments récupérés. En effet, dans la littérature, chaque phonème dure un certain temps (selon la langue, le plus petit dure un $20ms$ et le plus long, souvent voisé, dure $100ms$). Dans notre cas, on suppose que notre algorithme reconnait un ensemble de phonème, soit une syllabe. Dans ce cas, on considère donc qu'en dessous de $80ms$, on rejette le segment considéré. (LITTERATURE -REF ?)

Finalité Nous obtenons donc de l'acquisition d'un signal de trois secondes (choix arbitraires), un ensemble de segments contenant de la parole dans la majeur partie des cas (en effet, des bruits peuvent-être présents). Nous conclurons sur les manières d'améliorer notre algorithme. Néanmoins, cela ralentira notre algorithme sachant qu'il est déjà plus lent que ceux utilisés dans la littérature (on travaille avec des transformées de Fourier sur $257points$ pour chaque trame de $15ms$. La complexité des calculs est donc élevée). Nous allons développer par la suite les méthodes utilisées pour les différentes phases de l'algorithme de Détection d'activité vocale (on rappelle qu'il est composé d'un ensemble de sous-algorithmes).

3.2 La distance ou corrélation entre des vecteurs de bruit de fond et celui de chaque trame

L'article de Hongzhi Wang et Yuchao Xu, Meijing Li propose d'extraire les Mel Features Cepstral Coefficients (MFCCs) d'un signal vocal pour chaque trame en considérant les dix (10) premières trames comme des trames de bruit d'arrière plan. En réalisant la moyenne de ces vecteurs, on obtient un vecteur de $MFCC_{bruit}$. Ce vecteur va permettre de réaliser la Corrélation des MFCC de chaque nouvelle trame avec lui même (dans l'article, ils utilisent aussi la distance euclidienne, qui se révèle être moins performante).Le vecteur de $MFCC_{bruit}$ est mis à jour à chaque trame :

$$\underline{cno} = \underline{c}.p + (1-p).\underline{c} \tag{11}$$

où $\underline{cno}$ est le vecteur de $MFCC_{bruit}$, c sont les coefficients de chaque nouvelle trame et p est proche de 1. Le fait que p doit-être proche de 1 permet de garantir que le vecteur de MFCC bruité reste constitué de bruit de fond principalement malgré l'introduction de coefficients de parole à chaque nouvelle trame.

Nous nous sommes donc appuyé de ces travaux pour construire notre algorithme de détection vocale. Nous avons aussi appliqué le principe de corrélation à l'entropie spectrale, toujours dans le soucis de rendre notre algorithme le plus robuste au bruit.

3.3 Détermination des seuils

La détermination des seuils a été la partie la plus difficile à mettre en oeuvre, non pas que les algorithmes sont complexes, mais qu'une mauvaise méthode entraine la suppression des trames voulues. Ma première idée fut d'utiliser la distribution des valeurs obtenues. Cette distribution étant obtenue après avoir supprimé toutes les valeurs au-dessus de la moyenne du signal. Le but étant de récupérer la plus petites variations possibles du signal après analyse (Entropie et/ou MFCCs). Néanmoins, cette méthode bien qu'efficace était constante au court du temps ou du moins sur l'analyse d'un buffer d'échantillon. Il y avait donc des pertes et l'introduction de bruit après seuillage. La deuxième idée fut de trouver une méthode adaptée au chacun des paramètres utilisés.

3.3.1 Seuil pour les MFCCs

Concernant les MFCCs, je me suis appuyé d'une fonction sigmoïde mise à jour à chaque fenêtre. La sigmoïde a été améliorée afin d'avoir pour $x = 0$, une valeur proche de la valeur maximale du bruit (contrairement à la sigmoïde habituelle qui à pour valeur en $x = 0$, 0,5). Cette valeur maximale du bruit est généralement proche de zéro de part la méthode utilisée pour déterminer la distance entre des trames de bruit de fond et la trame en cours (la distance d'un vecteur de bruit de fond à un autre vecteur de bruit donne généralement un résultat faible).

3.3.2 Seuil pour l'entropie spectrale

La fonction de seuillage concernant l'entropie spectrale est quant à elle très standard du fait que les trames de parole sont très différentes de celle où il y a des silences ou du bruit. Pour cela, nous utilisons les 20 premières trames comme référence pour réaliser la valeur du seuil à l'instant 0. On considère la moyenne de ces 20 trames et on y ajoutant trois (3) fois l'écart type de ces 20 trames :

$$th_{noise} = \frac{1}{N}\sum_{k=1}^{N} x_k + 3.\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}|x_i - \mu|} \tag{12}$$

où μ représente la moyenne du signal considéré. En effet, on considère que sur les 20 premières trames ne représentent que bruit. En y ajoutant trois fois l'écart-type, on s'assure d'être au-dessus du bruit de fond dans sa majeur partie.
Pour avoir un seuil pertinent et précis, on réalise une mise à jour ce seuil. La mise à jour est effective dès que l'on a 10 trames à traiter. En effet, nous n'avons pas de pertinence à utiliser la moyenne et l'écart type sur une trame. Dix trames nous semblaient convenable pour pouvoir exploiter ces deux indices pour une mise à jour correcte de notre seuil. Les étapes de la mise à jour sont les suivantes :

- On collecte dix nouvelles trames
- On réalise l'opération suivante :

$$th_{new} = \frac{1}{N}\sum_{k=1}^{N} x_k + 3.\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}|x_i - \mu|} \tag{13}$$

- On met à jour la valeur du seuil pour les dix prochaines trames de la manière suivante (la mise à jour est similaire à celle du vecteur bruit pour l'entropie et les MFCCs) :

$$th = p.th_{noise} + (1-p)th_{new} \tag{14}$$

  Comme durant la mise à jour du vecteur bruit, on cherche à avoir une valeur de p proche de 1 (un) afin que la mise à jour ne soit pas polluée par une trame de voix. Cela rendrait nos seuil inutile car il ne considérerait plus les trames de voix ou alors celle avec des intensités plus élevées que celle considérée.

Amélioration possible du seuil On pourrait, tout comme le seuil des MFCCs, considérer une fonction sigmoïde particulière pour calculer le seuil optimal à chaque trame. Pour le moment, celle-là n'a pas été développée car les résultats étaient satisfaisants.

3.3.3 Segmentation et correction des labels

Nous en avions un peu parlé plus haut, une fois nos seuils identifiés, il faut labéliser chaque trame. Chaque label est soit un "1" soit un "0" ("0" correspondant à une trame sous le seuil). On réalise cette opération pour les deux paramètres que nous utilisions. Nous combinons nos résultats obtenus pour labéliser chaque trame (nous avons donc un "et" logique entre les deux vecteurs de labels obtenus).

Correction des "uns" et "zéros" isolés Notre vecteur de labels récupéré, ils arrivent que des "zéros" et des "uns" se retrouvent isolés dans une combinaison, respectivement, de "uns" et des "zéros". Afin de corriger ces "erreurs", on va chercher à évaluer le poids des labels entourant le label considéré. En soit, on balaye tous les labels afin de tous les vérifier. Les poids des labels entourant chaque label doit-être paire pour éviter des générer des nouvelles erreurs ou déplacer des labels (on a donc un nombre impaire des labels à évaluer pour trancher sur la valeur du label que l'on considère). On évalue donc le nombre de "un" et de "zéro", ce qui détermine si notre label est un "un" ou un "zéro". Mes propos ci-dessus n'étant pas forcément claire, illustrons mes propos avec un exemple :

1. Considérons un vecteur de labels : [1 1 0 0 1 1 1] et poids de 3 labels de part et d'autres du label que l'on évalue. On cherche à savoir si les "zéros" de ce vecteur de "uns" ne sont pas des erreurs. On se positionne sur le premier "zéro", on prend 3 labels à gauche et à droite (les premières valeurs n'ayant pas ou peu de labels sur leur gauche, on ne considère que le poids à d'autres en y intégrant au fur et à mesure les labels à gauche). On évalue donc le vecteur : [1 1 0 0 1 1]. On constate qu'il y a 4 "un" pour 2 "zéro". Le label "zéro" considéré est en faite un "un" (car il y a plus de poids de "un" que de "zéro"). Finalement, en évaluant l'ensemble du vecteur de label, on a : [1 1 1 1 1 1 1]. Notre vecteur est donc corrigé.

Aucune littérature n'a été utilisée pour traiter de ce problème, simplement des constats sur un grand nombre d'exemple. Le résultat obtenu est concluant, même si des améliorations sont tout à fait envisageable. Notamment sur le poids à considérer qui pourrait ne pas être fixe par exemple. On pourrait considérer que le poids devient plus grand quand, pendant un lapse de labels, ces derniers sont inchangeants. De fait, on pourrait corriger des "zéros"' ou "uns" isolés sur un plus grand poids.

Segmentation [1] Une fois le vecteur de labels corrigé, il nous reste plus qu'à extraire les segments où il y a de l'activité vocale. Pour cela, on considère un "head" et "tail". Le "head" caractérise le premier label "un" qui précède un "zéro" du vecteur et le "tail", le dernier "un" qui précède un groupe de "un" et est suivi d'un "zéro". Il peut y avoir plusieurs "head" et "tail" sur le signal acquisitionné. Le "head" est le premier marqueur, le "tail" lui n'apparait dès lors qu'un "head" est marqué sur le vecteur de labels. On extrait pour chaque "head" et "tail", leurs limites (la valeur de leur échantillon). Par exemple :

1. On considère un vecteur de label : [0 0 0 0 1 1 1 1 0 0 0] (**nb :** la taille du vecteur est plus conséquente dans l'algorithme). On balaye toutes les valeurs du vecteur jusqu'au premier "head" (un "un" qui est précédé d'un "zéro"). On place donc le "head" sur le "un" qui représente le cinquième échantillon de notre vecteur ([0 0 0 0 1 1 1 1 0 0 0], ici en rouge). Puis on continue l'analyse mais cette fois-ci pour trouver le "tail". Ici le "tail" est le "un" qui représente le huitième échantillon du vecteur ([0 0 0 0 1 1 1 1 0 0 0], ici en bleu). Le "head" et le "tail" délimite donc le segment suivant : [0 0 0 0 1 1 1 1 0 0 0].

1. L'idée s'appuit sur le principe des bufferisation que l'on utilise couramment en temps réel pour stocker nos données que l'on acquisitionne. Notamment en ce qui concerne les appelations "head" et "tail"

Ensuite, le "head" et le "tail" renvoient leurs limites (à savoir la valeur de leur échantillon). On obtient donc :

$$\begin{bmatrix} Limits(head) \\ Limits(tail) \end{bmatrix} = \begin{bmatrix} 5 \\ 8 \end{bmatrix}$$

Pour extraire notre vecteur, il nous reste plus qu'à réaliser la conversion : Trame $\Leftrightarrow$ Samples. Pour cela, on utilise la formule suivante pour calculer l'échantillon du "head" :

$$Sample = Limits(head).step_{ms} \tag{15}$$

où $step_{ms}$ représente la durée en ms de la fenêtre d'étude en ms moins le recouvrement que l'on considère en ms. Dans notre cas, elle est de $5ms$. Pour calculer l'échantillon du "tail", l'opération est similaire, mais il faut veiller à rajouter la longueur en ms de la fenêtre d'étude :

$$Sample = Limits(head).step_{ms} + window_{ms} \tag{16}$$

où $window_{ms}$ est la fenêtre d'étude considérée.

Les échantillons des "heads" et "tails" sont renvoyés dans un vecteur que l'on concatenate [2] avec ceux déjà retournés. On obtient finalement une matrice. On n'a plus qu'à extraire les segments en s'appuyant de la matrice précédemment trouvée pour l'ensemble du signal.

Post traitement des segments Nous avions déjà commencé à développer l'idée un peu plus haut, nous allons continuer ici. Ce qu'il faut retenir c'est que malgré notre correction des labels un peu plus haut, des erreurs subsistent. Vous nous direz : "Mais alors quel fut l'avantage de corriger les labels un peu plus haut ?". Tout simplement, la fonction de correction des labels permet de corriger des labels isolés. Néanmoins, elle ne prend pas en compte le fait que les segments doivent-être d'une certaine taille pour pouvoir être assimilé à une activité vocale. En effet, la durée est quelque chose que nous n'avions pas évoqué, mais c'est un paramètre important à prendre en compte. Un phonème peut durer entre $20ms$ et $100ms$, une syllabe $100ms$ à minima. Aucune contrainte n'a été faite sur la longueur (en sample) de nos segments. Il se peut que ceux-ci ne durent qu'une centaine d'échantillons ($i.e$ moins de 5ms à 16 $000Hz$). Il faut donc les rejeter car ne caractérise par une activité vocale. Pour cela, nous avons choisi de rejeter tous les segments représentants moins de $50ms$ en estimant qu'on ne peut avoir une activité vocale dans ce lapse de temps. Les autres segments sont quant à eux conservés.

3.4 Conclusion

Après avoir discuté des nos choix méthodiques et les algorithme mis en oeuvre, nous allons maintenant discuter des performances de notre algorithme de détection d'activité vocale en le comparant à des algorithmes.

2. Mots anglais : enchainement des vecteurs en soit

4 Tests et comparaisons

Dans cette partie, nous allons comparer notre algorithme à ceux de la littérature. La plupart des algorithmes ont été récupérés sur la plate-forme collaborative de Matlab en se basant sur leur cotation mais aussi le respect de la littérature.

4.1 Présentation des algorithmes évalués

Nous allons évaluer différents algorithmes à savoir :

- Un algorithme basé sur la détection de l'énergie et du taux de passage par zéro. Les seuils sont fixés de manière manuelle dans une plage prédéfinie [10].
- Un algorithme basé sur la détection de l'énergie ainsi que le centroïde spectrale ("centre de gravité").[5] Les seuils sont définis par détection des maximas locaux de leur histogramme. Pour des explications un peu plus précise, nous vous invitons à lire son article[6].
- Un algorithme basé sur la détection de l'entropie spectrale (dont nous avons parlé un peu plus haut). Le seuil est défini par une formule liant la moyenne à l'écart-type. Cet algorithme a été réalisé par mes soins en s'inspirant des plusieurs articles. [8][9]. Cet algorithme utilise des fonctions communes à notre algorithme.

Plusieurs tests seront opérés :

- Performances : Rapidité de la détection de l'algorithme.
- Robustesse au bruit : nous chercherons à ajouter progressivement du bruit dans nos tests en baissant le SNR (Signal Noise Ratio [3]).

Nous savons que nous partons dès le départ avec un désavantage. En effet, nous utilisons que des méthodes fréquentielles qui ont une complexité de calcul élevé. De plus, notre algorithme n'est pas optimisé car nous répétons certaines opérations plusieurs fois, notamment la plus gourmande, la Discrete Fourier Transform [4].

4.2 Robustesse au bruit

Notre première tâche avant de parler de performances en terme de temps d'exécution est de tester leur robustesse face aux bruits. En d'autres termes, sachant que nous allons travailler dans des environnements différents, il faut veiller à ce que l'environnement de travail n'empiète pas sur la reconnaissance vocale. Pour cela, nous allons réaliser plusieurs test sur un fichier audio contenant de la voix. On veillera dans un premier à ce qu'il est pas ou peu de bruit de fond. Puis nous en rajouterons progressivement. Enfin, le test ultime sera dans un environnement musical, nous mettrons en fond, une trame musicale (disons d'un style "métal" [5]). Le but de chaque algorithme est sur cinq tests [6] d'avoir le plus de trame avec présence vocale et le moins avec du bruit environnant.

3. Taux du rapport signal à bruit
4. Transformée de Fourier Discrète
5. Le style métal ne représente par forcément le style englobant du morceau que nous choisirons. Néanmoins, nous vous donnons le style métal pour que vous puissiez vous représenter le bruit généré par ce style de musique
6. Le nombre de tests est pris aléatoirement et de manière impaire

4.2.1 Données des tests

Les données suivantes seront utilisées pour tous les test. En cas de changement, nous les rappelerons :

1. Fréquence d'échantillonnage : 16 kHz
2. Quantification du signal : 16 $bits$
3. Taille des fenêtres : 15 ms
4. Décalage entre les fenêtres : 5 ms

4.2.2 Signal avec présence de voix et peu ou pas de bruit (SNR élévé)

Pour commencer, nous allons illustrer le signal que nous allons utiliser. Il est assez simpliste, dure environ 5s et dans un premier temps, le SNR est très élevé. La figure ci-dessous (Figure 1) est la représentation temporelle du signal avec des marqueurs indiquant la présence des mots "mode", "un et "deux". À titre d'information, le signal est échantillonné à 16kHz et quantitifé sur 16$bits$.

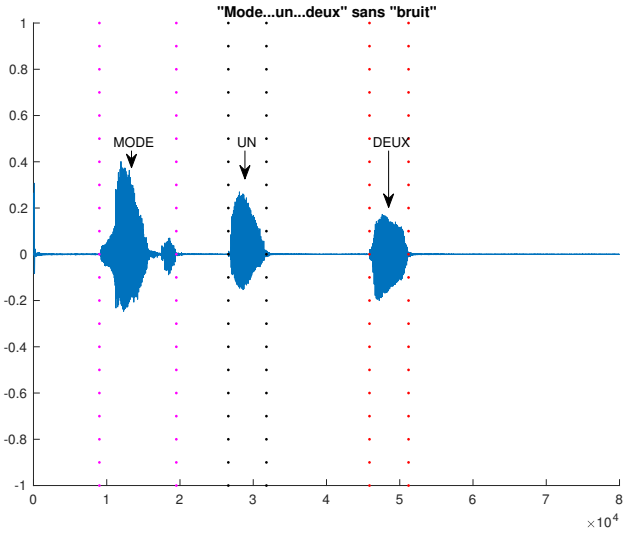

Figure 1 – Représentation temporelle du signal utilisé

Nous allons maintenant tester ce signal avec l'algorithme que nous proposons et ceux que nous avons soumis à titre comparatif.

Algorithme utilisant le ZRC (Zero Crossing Rate) et le STE(Short time Energy) [7]

Algorithme utilisant le Spectral centroid et le STE(Short time Energy) Après avoir utilisé notre "morceau" en argument, nous obtenons le résultat suivant : Les figures de l'algorithme n'étant pas forcément concrète sur le découpage (sauf si l'on sait que la partie grisée est la partie rejetée et les parties bordeaux sont celles conservées), nous avons utilisé les données retournées par l'algorithme pour vous présenter une figure similaire à Figure 1.

7. ZRC : taux de passage par zéro ; STE : énergie à court-terme

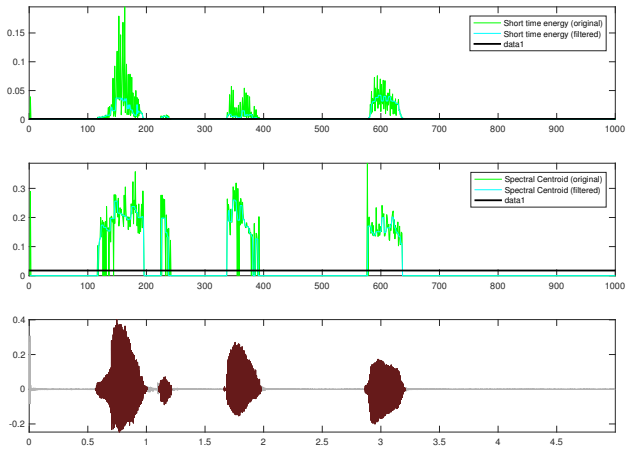

Figure 2 – Résultat de l'algorithme testé avec notre "morceau"

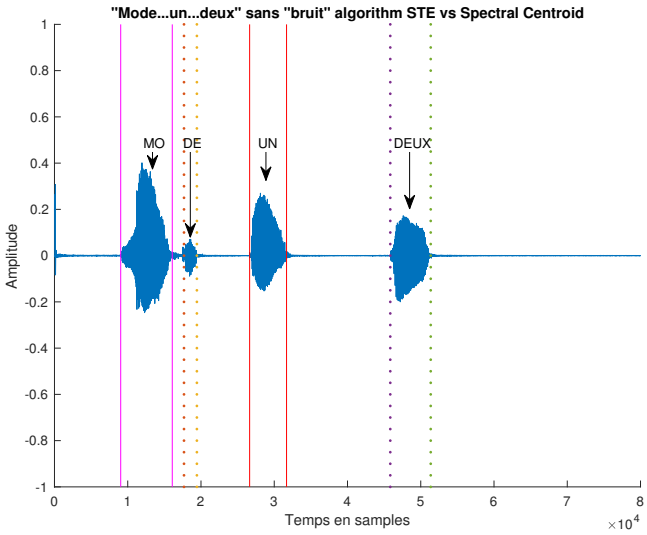

Figure 3 – Résultat de l'algorithme testé avec notre "morceau"

Etant donné que le découpage des segments n'était pas le même que celui présenté, nous avons labellisé les segments pour plus de compréhension. Comme vous pouvez le constater, le mot "mode" a été décomposé en syllabes. Ceci est dû au fenêtrage et au décalage très réduits.

Algorithme utilisant l'entropie spectrale En utilisant l'entropie spectrale avec un seuil définit ainsi :

$$threshold = \frac{1}{N}\sum_{k=1}^{N} x_k + 3.\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}|x_i - \mu|} \tag{17}$$

où N désigne le nombre d'éléments considérés (dans notre cas, les 10 premières trames supposés être du bruit), on obtient la segmentation suivante :

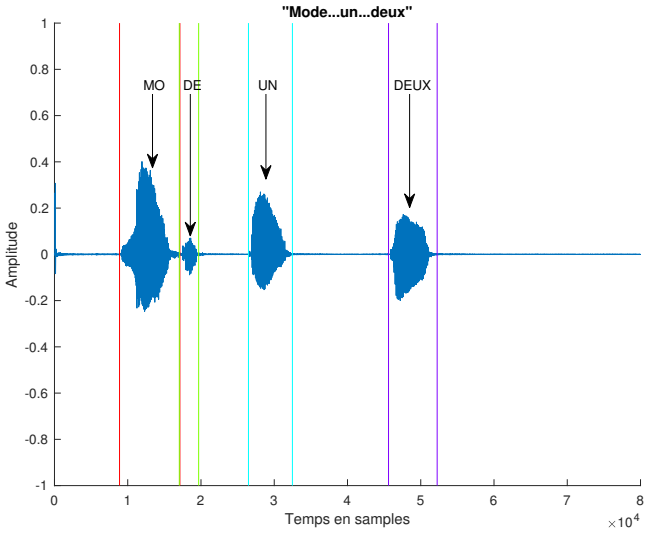

Figure 4 – Segmentation de notre morceau

On peut aisément constater (Figure 4) qu'il n'y a aucune erreur de segmentation même si les segments sont plus larges que ceux l'algorithme précédent. En s'appuyant de la Figure 5, nous observons que l'entropie spectrale dessine parfaitement l'activité vocale.

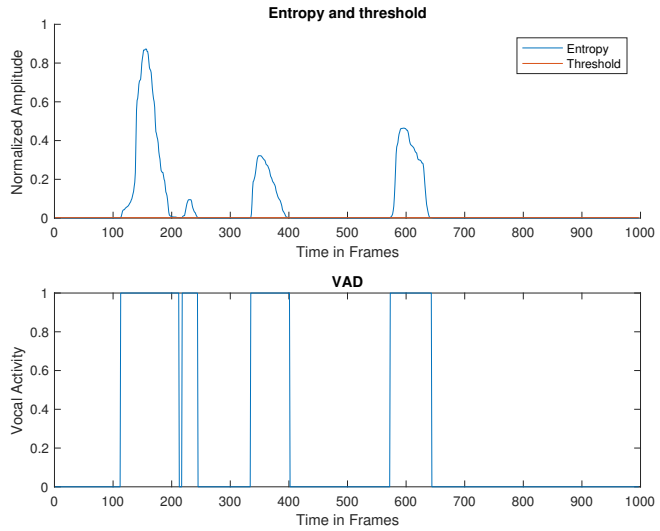

Figure 5 – Entropie et threshold

Notre algorithme Nous avions évoqué un peu plus haut, le fait que nous utilisons un signal dit de bruit de fond. Dans notre cas, nous allons utiliser la fin du signal (où il n'y a aucune présence vocale à des fins de vecteur de bruit). Dans les conditions normales d'utilisation, ce dernier aurait été directement acquis en amont du traitement. Notre signal servant de référence est réalisé de manière à avoir 1 seconde d'acquisition. Les courbes suivantes sont les résultats de nos différents paramètres utilisés. Les paramètres sont accompagnés de leur seuil. La dernière courbe symbolise notre vecteur de labels. Comme précédemment, nous allons symbolisé les différents morceaux comme nous l'avons fait

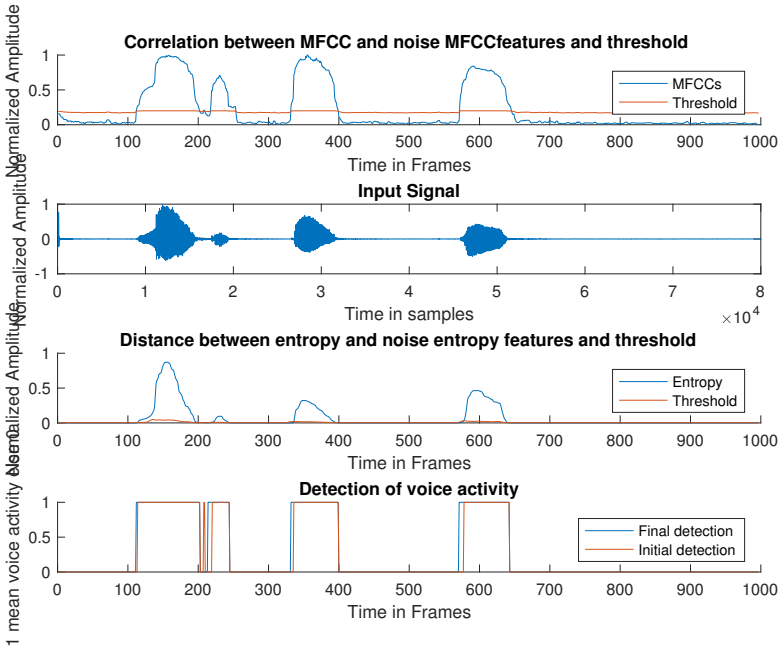

Figure 6 – Paramètres de détection de notre algorithme

tout au long du rapport afin d'en simplifier la lecture. Néanmoins, vous pouvez constater une détection de segments assez fiable. On remarque aussi que le mot "Mode" (cf Figure 1) en deux syllabes. Il est vrai que les légendes des figures selon "x" ne sont pas toutes les mêmes, néanmoins, on conçoit un alignement.

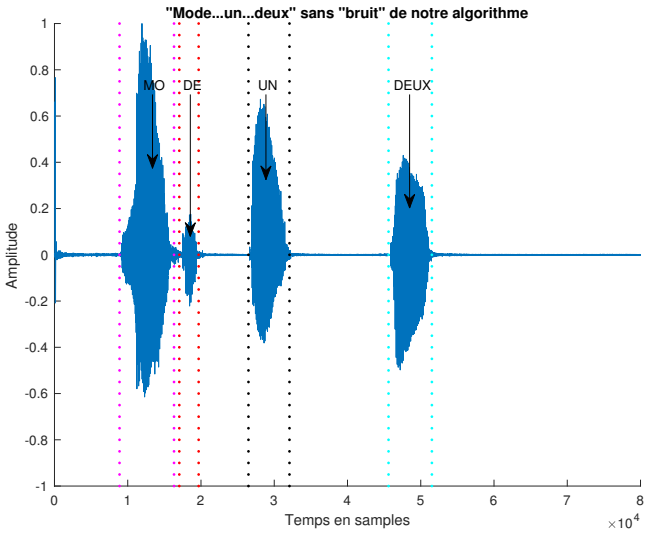

Figure 7 – Segmentation du signal par notre algorithme

4.2.3 Signal avec un SNR de $-30dB$

Nous remarquons que dans des conditions optimales, l'ensemble des algorithmes est capable de détecter les présences vocales du signal. Nous allons donc rajouter du bruit pour voir comment les algorithmes se comportent avec différents SNR[8]. Tous les SNR sont calculés à l'aide de la fonction Matlab *snr.m*. Le bruit est un bruit blanc gaussien généré à l'aide de la fonction *wgn.m* de Matlab. On obtient donc la représentation temporelle suivante :

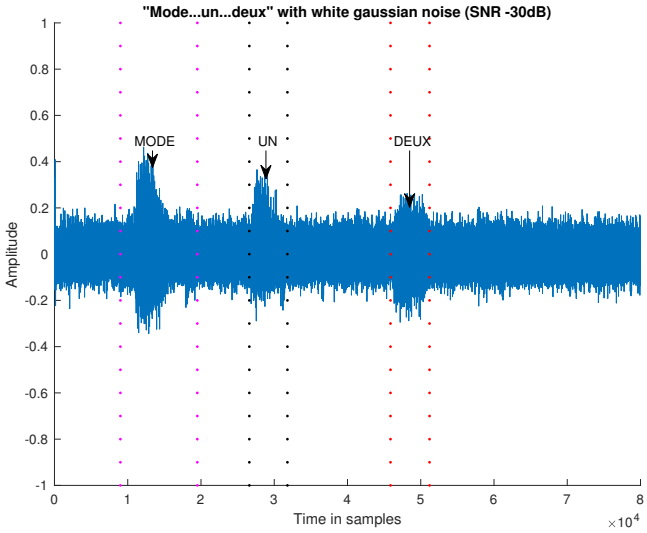

Figure 8 – Signal avec bruit blanc gaussien

8. Signal Noise Ratio : Rapport de Signal à Bruit

Algorithme utilisant le Spectral centroid et le STE(Short time Energy) Après avoir utilisé notre "morceau" bruité cette fois-ci en argument, nous obtenons le résultat suivant : La segmentation

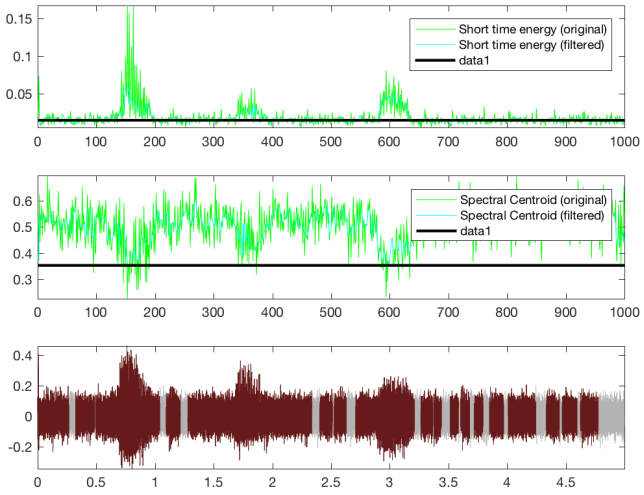

Figure 9 – Résultat de l'algorithme testé avec notre "morceau" bruité

du "morceau" bruité est la suivante. Vous pourrez assez vite constater que la largeur des segments à considérement augmentés. De même que l'introduction de nouveaux segments mais de "bruit".

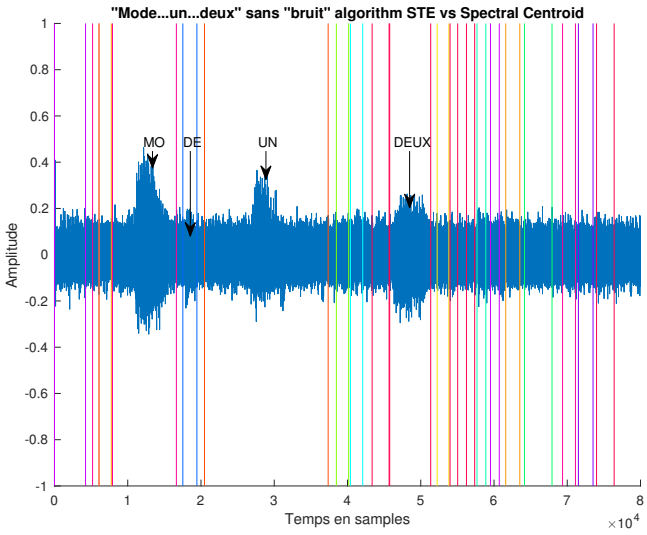

Figure 10 – Segmentation du "morceau" bruité avec marqueurs pour situer les mots

Il est vrai qu'il y a beaucoup de couleurs différentes et que la figure n'est pas forcément très lisible. néanmoins, afin de vous aider à vous repérer, vous pouvez toujours vous appuyez des segments grisés

et bordeaux de la Figure 9.[9]

Retour à notre algorithme Le bruit de fond est cette fois-ci un vecteur de bruit blanc gaussien d'une durée d'une seconde. Bien-sûr, il n'est en aucun cas un des tronçons de celui utilisé en addition à notre morceau. Nous obtenons donc le résultat suivant :

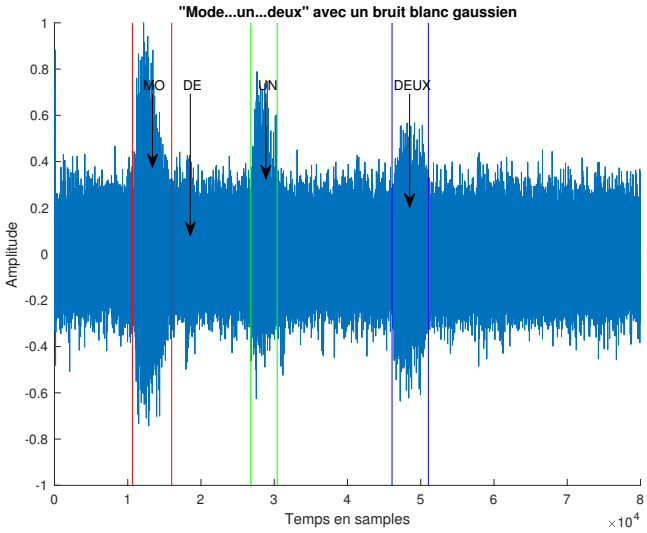

Figure 11 – Segmentation du "morceau" bruité avec marqueurs pour situer les mots

Vous pouvez constater que sur la Figure 11, nous n'avons aucun segment introduit par du bruit, néanmoins, la syllabe "de" du mot "mode" n'a pas été détectée. La question est pourquoi ? Pour cela, nous allons analyser les courbes de détection.

On remarque sur la courbe "Distance between entropy and noise entropy features and threshold", de fait que le seuil soit fixe, la syllabe "de" n'a pas été détectée. C'est sans doute ne amélioration que nous pourrions apporter par la suite à notre algorithme. De plus, sur la courbe "correlation between MFCC and noise MFCCs features", notre algorithme calculant la distance entre le vecteur bruit et le vecteur de MFCC que l'on calcule à chaque instant est un peu instable au début, malgré que l'on note la présence de "pics". Pour conclure sur cette simulation, nous pouvons déjà argumenté que notre algorithme est robuste au bruit même si on engendre un perte de syllabe dit "non plosives". Quelques améliorations et modifications pourraient assurer la détection de cette syllabe néanmoins. On reste satisfait de notre détection. En effet, un algorithme de détection d'activité vocale ne travaillant jamais seul, il rendra les algorithmes de reconnaissance de la parole plus performant. Le gain de temps à ne pas traiter du "bruit" est donc presque nulle pour notre algorithme comparé aux autres.

9. Les figures sont de même dimension sur ce rapport.

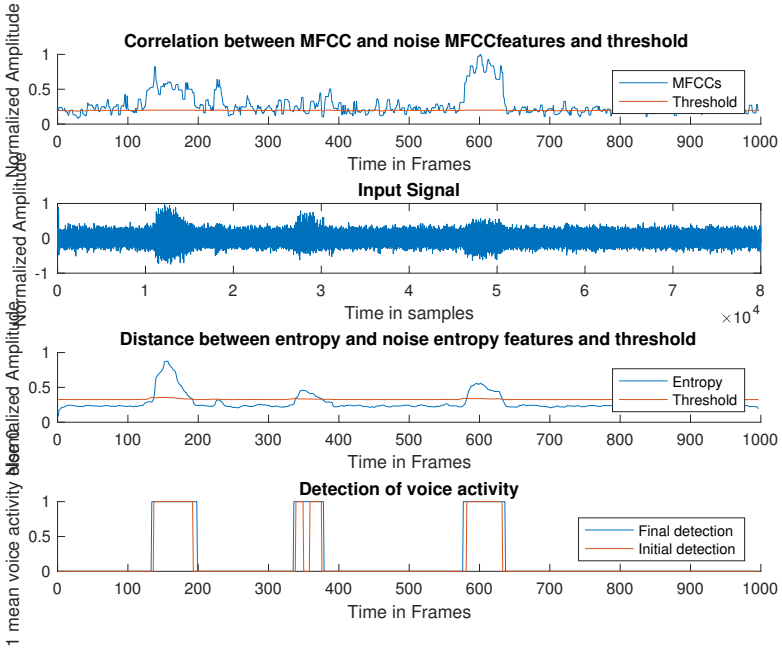

Figure 12 – Courbes de détection de notre algorithme

4.2.4 Signal avec un musique de fond - SNR -13dB

Nous aimons un peu la difficulté, alors nous allons utiliser, comme nous l'avons énoncé dans l'introduction un peu plus, une musique de style "métal" [10]. La musique utilisée est **Rammvier** de Rammstein [11]. Le choix de cette musique n'est pas arbitraire et pas seulement par goût. Le spectre de cette musique est assez densément rempli dans la bande passante que nous étudions. De plus, la musique est bien sûr remplie de parole (nous avons donc la présence d'une voix parasite). Le but de cette dernière expérience est donc de vérifier que notre algorithme est robuste contre le bruit musical et les voix dans des trames musicales (dans le cas général, on ne peut empêcher la détection de la voix si elle est bien plus présente que celui du locuteur).

La musique prélevée n'est néanmoins pas à une fréquence d'échantillonnage de 16 kHz. Il faut donc sous-échantillonner les morceaux choisis. Pour cela, afin d'éviter tout repliement du signal, on commence par appliqué un passe-bas à 8 kHz (théorème de Shannon). Puis, une fois cette opération réalisée, nous pouvons sous-échantillonner nos morceaux et les ajouter à notre "morceau".

Algorithme utilisant le Spectral centroid et le STE(Short time Energy) Les résultats sont exposés sur la Figure 13. Afin d'avoir un meilleur aperçu de la segmentation, nous vous donnons, comme précédemment le Signal "bordeaux" qui représente les zones où l'algorithme détecte une présence de voix et "grisé" les zones rejetées Figure 14.

10. La définition de style différant entre les individus, la musique n'est peut-être pas représentative du style "métal" mais d'un style en déclinant.
11. Pour nos tests nous avons simplement utilisé un échantillon de 5s et une autre seconde pour notre vecteur bruit.

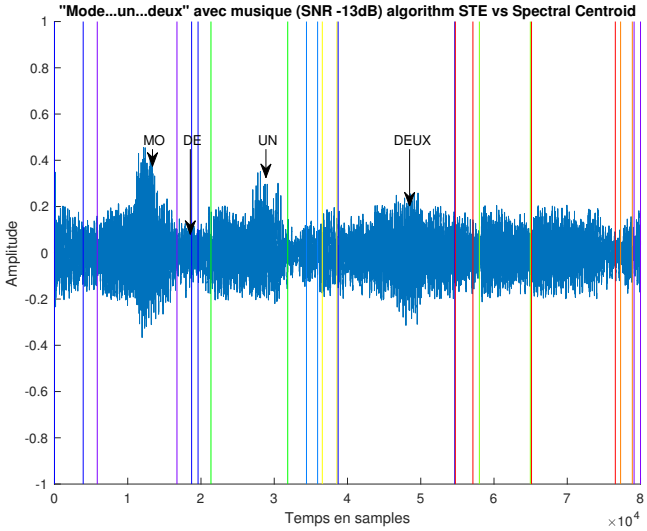

Figure 13 – Segmentation du Signal

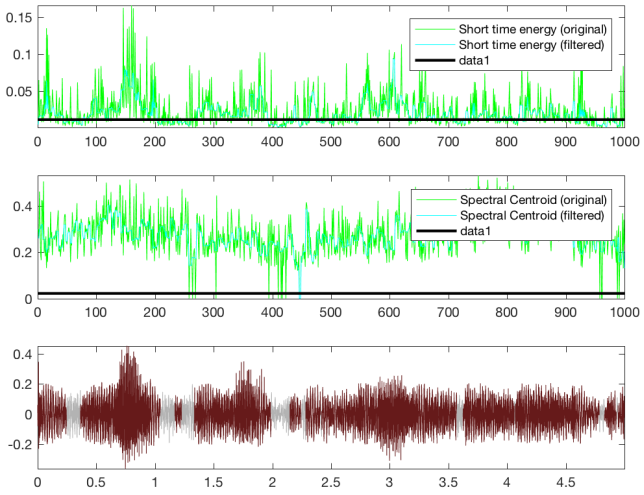

Figure 14 – Segmentation du Signal et courbes de détection

On remarque, comme précédemment, que dès qu'il y a un SNR (taux de signal à bruit) trop faible, l'algorithme renvoie beaucoup de segments où il n'y a que de "bruit" et omet certains segments de voix, notamment "de" (du mot "mode"). On peut donc aisément dire que l'algorithme est peu robuste au bruit.

Notre algorithme Malheureusement pour nous, nous arrivons au même constat que précédemment. Même si nous détectons peu ou pas de "bruit", nous n'avons toujours pas le segment contenant le phonème plosive "de" Figure 15. Nous avons tout de même l'introduction d'un segment de bruit, ce

qui n'est pas considérable.

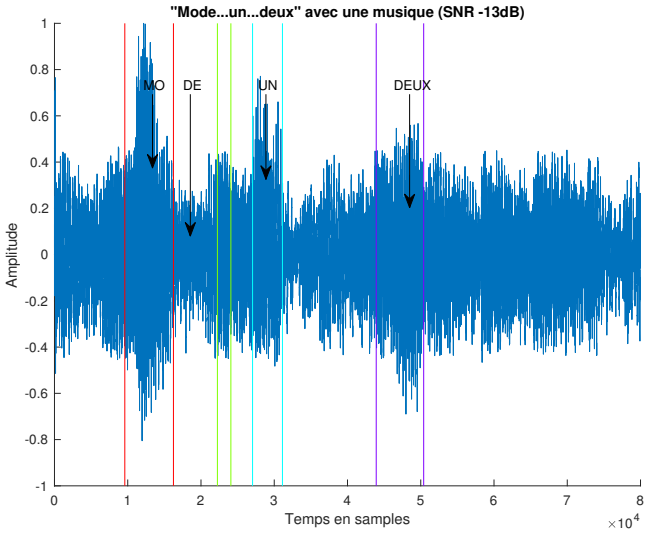

Figure 15 – Segmentation du Signal

4.2.5 Conclusion

On peut conclure cette partie concernant la robustesse aux bruits en énonçant que notre algorithme est très robuste au bruit comparé à ceux présentés. On voit très rapidement que le bruit "parasite" la détection de segments de voix. Ce n'est pas le cas pour notre algorithme même si des phonèmes ne sont pas détectés. Il faut donc par la suite veiller à bien superviser notre algorithme de reconnaissance de la parole [12]. En d'autres termes, il devra considérer des segments manquants selon le bruit qu'il détecte ou non. De plus, comme notre algorithme détecte des segments de voix (à quelques erreurs près), nos algorithmes en aval pourront donner des résultats plus précis. En effet, ils ne seront pas pollués par des données erronées (ou contenant de "bruit" qui n'ont aucune utilités).

Après avoir veillé à respecter le cahier des charges concernant la robustesse au bruit, il s'agit de nous intéresser aux performances de notre algorithme.

12. On parle ici de l'algorithme global et non le VAD présenté ici.

4.3 Performances

Nous en avons parlé un peu plus haut, mais outre la robustesse, il faut être performant. Rien ne sert d'avoir un algorithme qui fait peu d'erreur mais qui prend cent fois plus de temps. Il est donc important de comparer la vitesse d'exécution des algorithmes proposés au notre. Bien-sûr, on sait d'or-et-déjà que notre algorithme est plus lent que ceux présentés. Comment le savoir avant même de réaliser les tests de chronos ? Tout simplement car nos méthodes n'emploient que des caractéristiques fréquentielles. De part la complexité des calculs engendrés, nous n'aurons donc de meilleurs temps d'exécution. Néanmoins, si nous sommes en dessous de vingt fois leur temps, nous pourrons considérer notre algorithme performant au vue de sa robustesse au bruit. Nous générons en effet moins de segments parasites qui ralentiront le traitement par la suite. Enfin, nous savon que notre algorithme est grandement améliorable notamment en terme de répétition. Nous avons, après analyse, constaté que nous répétions certaines opérations, notamment les transformées de Fourier Discrète plusieurs fois (quatre fois dans notre cas, deux pour le calcul de l'entropie, deux autres pour les MFCCs). Nous pourrions gagner un temps considérable. L'algorithme évoluera à cet effet lorsque nous l'implémenterons en C++.

4.3.1 Déroulement

Pour quantifier les performances des algorithmes que nous confrontons au notre, nous avons choisi d'utiliser les fonctions "tic" et "toc" sur Matlab®. Nous avons réaliser plusieurs fois les algorithmes afin d'en relever le temps moyen d'exécution. Ils seront résumés dans le tableau un peu plus bas. On considère le temps de départ ("tic") après que l'on est acquis le signal que l'on souhaite étudier. Le temps de fin ("toc") est positionné à la fin de l'exécution de la dernière fois des algorithmes. De plus, le temps d'exécution est calculé pour les différents cas proposés ci-dessus.

4.3.2 Résultat

Nous nous tâcherons de commenter les résultats après les avoir exposés ci-dessous.

Algorithme	Temps moyen	Temps min	Temps max	Ratio
Notre algorithme	0.802	0.7708	0.8803	41
Algorithme utilisant le Spectral centroid et le STE	0.0195	0.0184	0.0216	1

Nous ne sommes pas peu fier des performances de notre algorithme même si le temps d'exécution pour des trames de $5s$ est assez conséquent (40 fois plus lent que le plus rapide). Néanmoins, nous pouvons au moins réduire ce temps de moitié si ce n'est plus. Enfin, nous venons de mener une opération de détection sur une trame en temps non-réel. Peut-être est-il plus performant en temps réel ? Nous veillerons lors de notre implémentation en C++ à réduire le temps d'exécution de l'algorithme que nous vous avons présenté. Nous pensons qu'un réduction au tiers serait un bel enjeu. Cela permettrait de se qualifier de performant au vue de la complexité des calculs que nous employons. Une optimisation temps réel est aussi possible.

Références

[1] Mel frequency cepstral coefficient (mfcc) tutorial. `http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/`. [Online ; Accessed : 2017-03-15].

[2] Record and play audio. `https://www.mathworks.com/help/matlab/import_export/record-and-play-audio.html`. [Online ; Accessed : 2017-02-04].

[3] Ouzounov .A. Robust features for speech detection a comparative study. *In Int. Conference on Computer System and Technology*, 2005.

[4] Patrick Durand and René Durand. Les tomates tueuses. *Le beau journal*, page 24, jan 2007.

[5] Theodoros Giannakopoulos. Silence removal in speech signals. `https://www.mathworks.com/matlabcentral/fileexchange/28826-silence-removal-in-speech-signals`. [Online ; Accessed : 2017-03-27].

[6] Theodoros Giannakopoulos. A method for silence removal and segmentation of speech signals, implemented in matlab. 2009.

[7] Faran Awais Butt Madiha Jalil and Ahmed Malik. Short-time energy, magnitude, zero crossing rate and autocorrelation measurement for discriminating voiced and unvoiced segments of speech signals. *978-1-4673-5613-8©2013 IEEE*, Mai 2013.

[8] Philippe Renevey and Andrzej Drygajlo. Entropy based voice activity detection in very noisy conditions. *In Int. Conference on Computer System and Technology.* [Online ; Accessed : 2017-03-27].

[9] Hung J. Shen J and Lee J. Robust entropy-based endpoint detection for speech recognition in noisy environments. *In Fifth International conference on spoken Language Processing*, 1998.

[10] Prof. Lawrence Rabiner (Rutgers University, Prof. Ronald Schafer (Stanford University) Kirty Vedula University of California, Santa Barbara), and Siva Yedithi (Rutgers University). Endpoint detector. `https://www.mathworks.com/matlabcentral/fileexchange/45314-endpoint-detectorl`. [Online ; Accessed : 2017-03-27].

[11] Hongzhi Wang and Meijing Li Yuchao Xu. Study on the mfcc similarity-based voice activity detection algorithm. *Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), 2011 2nd International Conference on*, 2011.