Système de détection de mouvements complexes de la main à partir des signaux EMG, pour le contrôle d'une prothèse myoélectrique

Mémoire

Roxane Crepin

Maîtrise en génie électrique - avec mémoire
Maître ès sciences (M. Sc.)

Québec, Canada

Système de détection de mouvements complexes de la main à partir des signaux EMG, pour le contrôle d'une prothèse myoélectrique

Mémoire

Roxane Crepin

Sous la direction de :

Benoit Gosselin, directeur de recherche
Alexandre Campeau-Lecours, codirecteur de recherche

Résumé :

Les avancées technologiques en ingénierie biomédicale à travers le monde permettent le développement de systèmes automatisés et adaptés, visant à fournir aux personnes vivant avec un handicap un meilleur confort de vie. Les prothèses intelligentes basées sur l'activité myoélectrique permettent aux personnes amputées d'interagir intuitivement avec leur environnement et d'effectuer des activités de la vie quotidienne. Des électrodes placées sur la surface de la peau et une électronique embarquée dédiée recueillent les signaux musculaires et les traduisent en commandes pour piloter les actionneurs de la prothèse.

Atteindre une performance accrue tout en diminuant le coût des prothèses myoélectriques est une étape importante dans l'ingénierie de réadaptation. Les mains prothétiques, actuellement disponibles à travers le monde, bénéficieraient d'un contrôle plus efficace et plus intuitif. Ce mémoire présente une approche en temps réel pour classifier les mouvements des doigts à l'aide des signaux d'électromyographie (EMG) de surface. Une plateforme multicanale d'acquisition de signaux, de notre conception, est utilisée pour enregistrer 7 canaux EMG provenant de l'avant-bras. La classification des signaux EMG est effectuée en temps réel, en utilisant une approche d'analyse discriminante linéaire. Treize mouvements de la main peuvent être identifiés avec une précision allant jusqu'à 95,8% et de 92,7% en moyenne pour 8 participants, avec une prédiction mise à jour toutes les 192 ms. L'approche a voulu être adaptée pour créer un système embarqué ouvrant de grandes opportunités pour le développement des prothèses myoélectriques légères, peu coûteuses et plus intuitives.

Abstract:

Technological advances in biomedical engineering worldwide enable the development of automated and patient-friendly systems, aiming at providing the severely disabled a better comfort of life. Intelligent prostheses based on myoelectric activity allow amputees to intuitively interact with their environment and perform daily life activities. Electrodes placed on the surface of the skin, and dedicated embedded electronics allow to collect muscle signals and translate them into commands to drive a prosthesis actuators.

Increasing performance while decreasing the cost of surface electromyography (sEMG) prostheses is an important milestone in rehabilitation engineering. The prosthetic hands that are currently available to patients worldwide would benefit from more effective and intuitive control. This memoir presents a real-time approach to classify finger motions based on sEMG signals. A multichannel signal acquisition platform of our design is used to record forearm sEMG signals from 7 channels. sEMG pattern classification is performed in real time, using a Linear Discriminant Analysis (LDA) approach. Thirteen hand motions can be successfully identified with an accuracy of up to 95.8% and of 92.7% on average for 8 participants, with an updated prediction every 192 ms. The approach wanted to be adapted to create an embedded system opening great opportunities for the development of light-weight, inexpensive and more intuitive electromyographic hand prostheses.

Table des matières :

Résumé : ..... **iii**
Abstract: ..... **iv**
Liste des tableaux : ..... **vii**
Liste des figures : ..... **viii**
Liste des abréviations : ..... **x**
Remerciements : ..... **xi**

1 Introduction **1**
1.1 Contributions **1**
1.2 Objectifs du projet **2**
1.3 Organisation du mémoire **2**

2 État de l'art sur les prothèses myoélectriques **4**
2.1 Les prothèses myoélectriques commerciales **4**
2.2 Les problèmes rencontrés **9**
2.3 Les améliorations en développement **10**
2.3.1 Électrodes implantées 10
2.3.2 Les capteurs de force sensible 10
2.3.3 Le retour de sensation 10
2.3.4 Maintien automatisé des objets 12
2.3.5 Contrôle de la force et de la vitesse 13
2.3.6 Temps d'exécution de la prothèse 13
2.3.7 Contrôle simultané du poignet et de la main 14
2.3.8 Motorisation du coude à l'aide de câbles de position 14
2.3.9 Utilisation du mouvement de l'épaule pour contrôler la prothèse 15
2.3.10 Utilisation de l'humérus 15
2.4 Les méthodes de classification des signaux **16**
2.4.1 Méthode sans apprentissage préalable 16
2.4.2 L'intelligence artificielle 17
2.4.3 Comparaison de certaines méthodes par l'expérimentation 18
2.5 Les prototypes en développement **19**
2.6 Améliorations futures **22**
2.6.1 Avoir de meilleurs protocoles expérimentaux et analyser les attentes des usagers 22
2.6.2 Commande vocale 23
2.6.3 Contrôle par le mouvement des yeux 23
2.6.4 Informations audio 23
2.6.5 Utilisation des nerfs pour le retour de sensation 24

3 Vue globale technique du projet **25**

4 Les électrodes et leurs positions **27**
4.1 Les muscles de l'avant-bras **27**
4.2 Choix du type d'électrode **29**
4.3 Conception des électrodes **29**
4.4 Nombre d'électrodes utilisées et placement sur l'avant-bras **31**

5 Acquisition et traitement des signaux EMG **32**
5.1 Présentation générale des besoins **32**
5.2 Le microcontrôleur MSP430F5529 ..... **32**
5.3 1ers traitements sur platine d'expérimentation **33**

5.4 Remplacement de la platine d'expérimentation par l'ADS1298 35
5.5 Développement d'un PCB pour l'acquisition et le traitement des signaux EMG 37
5.6 Réception des données et traitement numérique des signaux EMG sous Matlab 37

6 Algorithme de classification des signaux EMG sous Matlab 40
6.1 Vue globale et objectifs 40
6.2 La méthode *Linear Discriminant Analysis* 40
6.3 Le choix des paramètres discriminants utilisés 42
6.4 Le découpage des fenêtres et la classification temps réel 44
6.5 Développement d'une interface graphique Matlab pour la construction de la base de données 46
6.6 Développement d'une interface graphique pour la prédiction en temps réel 48

7 Développement d'un système embarqué sur Raspberry Pi Zero 49
7.1 La Raspberry Pi Zero W 49
7.2 Développement du code Matlab de classification en C++ 50
7.2.1 Structure de l'algorithme 50
7.2.2 Librairies utilisées 51
7.2.3 Description du fichier principal 52
7.3 Portage du code sur Raspberry Pi Zero 54
7.4 Expérimentations sur la Raspberry Pi Zero 55

8 Design de la prothèse mécatronique 57
8.1 Main en impression 3D 57
8.2 Choix des moteurs, réducteurs et encodeurs 58
8.2.1 Type de moteur 58
8.2.2 Vérification des besoins 60
8.3 Choix des contrôleurs 62
8.4 Design final 63

9 Expérimentation 65
9.1 Protocole expérimental 65
9.2 Efficacité 66

10 Discussion et travaux futurs 70

11 Conclusion : 74

Bibliographie : 75
Annexe A : Code Matlab des interfaces graphiques (Partie 6) 81
Annexe B : Code C implémenté dans le microcontrôleur MSP430F5529 (Partie 5) 102
Annexe C : Code C++ implémenté sur Raspberry Pi Zero W (Partie 7) 109

Liste des tableaux :

Tableau 1 : Comparaisons des prothèses. 8
Tableau 2 : Exigences pouvant être attendues par une personne amputée. Tableau provenant de l'article [8]. 23
Tableau 3 : Tableau des performances du circuit. 34
Tableau 4 : Tableau récapitulatif des paramètres de l'ADS1298. 36
Tableau 5 : Forme de la matrice de la base de données de taille 10140x1050. 46
Tableau 6 : Avantages et Inconvénients de différents types de moteurs. 58
Tableau 7 : Prix du système développé. 64
Tableau 8 : Efficacités de classification (en %) obtenues pour chacun des participants et des classes (Fig 32) avec la fonction predict. 68

Liste des figures :

Figure 1 : Main sensor speed (Ottobock) [4]. 5
Figure 2 : Main Michelangelo (Ottobock) [4]. 5
Figure 3 : Main i-limb Ultra Revolution (Touch Bionics) [5]. 6
Figure 4 : Système i-limb Digits (Touch Bionics) [5]. 6
Figure 5 : Main Transcarpienne (Ottobock et Touch Bionics). 6
Figure 6 : Le greifer (Ottobock) [4]. 7
Figure 7 : Main bebionic3 (Ottobock) [6]. 7
Figure 8 : **(a)** Main d'enfant, **(b)** poignet Myo Wrist, **(c)** coude Dynamic Arm. 8
Figure 9 : Comparaison du signal qui traverse l'avant-bras humain et de celui provenant des prothèses myoélectriques modernes. Image provenant de l'article [8]. 11
Figure 10 : Modes de préhension principaux de la main : latéral **(a)**, cylindrique **(b)** et tripode **(c)**. Image provenant de l'article [8]. 13
Figure 11 : Résumé du chemin entre l'entrée et la sortie du système de la prothèse. Image provenant de l'article [8]. 14
Figure 12 : Signal brut et signal après application de la méthode TKE. Image provenant de l'article [27]. 17
Figure 13 : Prothèse MANUS Hand. 20
Figure 14 : Prothèse FluidHand. 21
Figure 15 : Prothèse Southampton Hand. 21
Figure 16 : Prothèse CyberHand. 21
Figure 17 : Schéma-bloc du système de détection des mouvements de la main. 25
Figure 18 : Placement des muscles extenseurs dans l'avant-bras [Teitarc.com]. 27
Figure 19 : Placement des muscles fléchisseurs dans l'avant-bras [nospot.org]. 28
Figure 20 : Placement des 7 électrodes sur l'avant-bras, plus une de référence. 28
Figure 21 : Électrodes avec pads en cuivre. 30
Figure 22 : Électrodes avec pads en or. 30
Figure 23 : Placement réel des électrodes sur l'avant-bras. 31
Figure 24 : Circuit réalisé pour un canal. 33
Figure 25 : Schéma bloc de l'acquisition par microprocesseur pour 2 canaux, lors de l'utilisation de la Breadboard. 34
Figure 26 : Courbe du signal provenant du canal annulaire en flexion en sortie de la Breadboard avec l'acquisition sous Matlab. 35
Figure 27 : Breadboard contenant 5 exemplaires du circuit de la Figure 24. 35
Figure 28 : ADS1298 de Texas Instruments USA. 36
Figure 29 : Carte d'acquisition et de traitement des signaux EMG (56 x 45 mm^2). 37
Figure 30 : Batterie LiPo. 37
Figure 31 : Courbe du signal provenant du canal du pouce en flexion en sortie de la carte d'acquisition et après filtrage sous Matlab. 39
Figure 32 : Les différents mouvements (classes) à reconnaître. 40
Figure 33 : Exemple de séparation de classes par la méthode LDA. 42
Figure 34 : Algorithme de classification. 45
Figure 35 : Interface graphique pour la construction de la database lors d'une flexion du pouce. 47
Figure 36 : Interface graphique de prédiction lors d'une flexion du pouce. 48

Figure 37 : Raspberry Pi Zero W. 49
Figure 38 : Vue générale du système. 56
Figure 39 : Main en impression 3D. 57
Figure 40 : **(a)** Moteur, **(b)** Réducteur et **(c)** Encodeur incrémental. 60
Figure 41 : Configuration d'un motoréducteur. 60
Figure 42 : ESCON Module 24/2 de chez Maxon. 62
Figure 43 : Interface graphique ESCON Studio de chez Maxon, pour le contrôleur. 63
Figure 44 : Design final de la prothèse mécatronique. 64
Figure 45 : Système de détection en temps réel au complet et interface graphique utilisée pour la construction de la database. 65
Figure 46 : **(a)** Un signal EMG, **(b)** la forme TKE du même signal, et **(c)** la Transformée de Fourier du signal, pour un mouvement de flexion du pouce sur le canal 1. 66
Figure 47 : Code Matlab utilisé pour les calculs d'efficacité avec predict. 67
Figure 48 : Pourcentage total d'efficacité de prédiction pour chacun des 8 participants. 68

Liste des abréviations :

ADC Analog to Digital Converter
ADS ADS1298
AFE Analog Front-End
AR Autorégressif
CIRRIS Centre Interdisciplinaire de Recherche en Réadaptation et Intégration Sociale
CIUSSS Centre intégré universitaire de santé et de services sociaux
CPU Central Processing Unit
CSP Common Spatial Patterns
DC Courant continu
EMG Électromyographie
FFT Fast Fourier Transform
FSR Force Sensitive Resistor
HA Hjorth Activity
HC Hjorth Complexity
HDMI High-Definition Multimedia Interface
HM Hjorth Mobility
IEMG Integrated EMG
IIR Infinite Impulse Response
kNN k-Nearest Neighbor
LDA Linear Discriminant Analysis
LiPo Lithium Polymère
MAV Mean Absolute Value
MNF Mean Frequency
MSP MSP430F5529
OS Operating System
PCB Printed Circuit Board
RAM Random Access Memory
RF Radio Fréquence
SD Secure Digital
sEMG surface electromyography
sinc sinus cardinal
SK Skewness
SNR Signal to Noise Ratio
SOA State Of the Art
SPI Serial Peripheral Interface
SVM Support Vector Machine
TKE Teager-Kaiser Energy
TMR Targeted Muscle Reinnervation
UART Universal Asynchronous Receiver-Transmitter
USB Universal Serial Bus
WAMP Wilson Amplitude
WL Waveform Length
WPT Wavelet Packet Transform

Remerciements :

Je tenais à remercier le personnel du programme des aides techniques en appareillage pour la clientèle adulte et aînée en déficience motrice, du CIUSSS (Centre intégré universitaire de santé et de services sociaux) de la Capitale-Nationale, sous la direction de Valérie Brodeur pour m'avoir fourni leurs expertises sur les prothèses myoélectriques. Je remercie également Christian Brideau, du département de réadaptation de la faculté de médecine à l'Université Laval, pour sa sympathie et pour m'avoir permis d'y voir plus clair sur les muscles de l'avant-bras. Mais également Ulysse Côté-Allard pour ses connaissances poussées sur les méthodes de classifications des signaux. Merci au Centre Interdisciplinaire de Recherche en Réadaptation et Intégration Sociale (CIRRIS), du CIUSSS de la Capitale-Nationale, pour avoir soutenu ce projet.

Je tiens à remercier les participants aux tests sur le système développé, et notamment pour le temps qu'ils ont pu me consacrer. Bien évidemment, je remercie toute l'équipe du laboratoire de microsystèmes biomédicaux de l'Université Laval pour avoir répondu à mes nombreuses questions.

J'ai également une pensée particulière pour Latyr Cheikh Fall et Quentin Mascret pour leurs supports au quotidien, leurs patiences et leurs aides précieuses sur ce projet. Vous avez su me soutenir et me rassurer dans les moments difficiles alors merci pour tout.

Enfin, je remercie chaleureusement mes deux codirecteurs de recherche : Benoit Gosselin, professeur et directeur du laboratoire de microsystèmes biomédicaux, et Alexandre Campeau-Lecours, professeur et membre du laboratoire de robotique de l'Université Laval. Merci pour vos conseils, vos expertises et vos encouragements tout au long de ce projet.

Sans vous tous, ce projet n'aurait pas pu aboutir, alors encore une fois merci !

1 Introduction

Il est difficile d'estimer de façon précise le nombre de personnes amputées, mais presque 2 millions [1] de personnes vivent avec une amputation, toutes confondues, aux États-Unis, et chaque année environ 185 000 amputations [2] ont lieu. Parmi les personnes vivant avec une amputation, les principales causes sont les maladies vasculaires (54%), y compris le diabète et les maladies artérielles périphériques, les traumatismes (45%) et le cancer (moins de 2%) [1]. De plus, on compte 4 fois plus d'amputations de membre inférieur que de membre supérieur [3]. Afin de répondre au nombre croissant de personnes amputées dans le monde, des prothèses myoélectriques, adaptées à tout type d'amputation, se développent de plus en plus sur le marché biomédical. Celles-ci se basent sur les contractions musculaires générées par l'usager pour se mettre en mouvement. Néanmoins, nous pourrons constater, dans la partie suivante sur l'état de l'art, que les fonctionnalités des prothèses myoélectriques restent limitées. En effet, certaines permettent d'accéder à peu de degrés de liberté pour les usagers, comme la main Sensor Speed ou le Greifer (Ottobock) [4]. D'autres, comme I-limb Ultra Revolution (Touch Bionics) [5] ou Bebionic3 (Ottobock) [6], doivent se paramétrer afin de sélectionner les mouvements voulus, ainsi que le nombre de contractions du muscle à produire pour activer un des mouvements préenregistrés. Ces prothèses commerciales ont l'avantage d'assurer une très bonne robustesse et possèdent un système de contrôle relativement simple. Malgré cela, un manque d'intuitivité peut très vite se faire ressentir. De plus, le prix des prothèses s'envole rapidement dû à l'utilisation de composants très coûteux.

Les utilisateurs cherchent avant tout des prothèses qui répondent au mieux à leurs besoins dans la vie de tous les jours, ainsi que des prothèses avec un contrôle simple et intuitif, qui permettent d'effectuer des mouvements précis et rapides. Des recherches, [7] [8], sont en cours afin de développer de nouveaux prototypes poussant les fonctionnalités des prothèses commerciales. La partie sur l'état de l'art permettra de nous donner un aperçu de ces recherches.

Ce projet devra faire face à de nombreux défis, du point de vue de l'instrumentation, de l'intelligence artificielle ou même de la mécanique. En effet, nous pourrons constater les difficultés de traitement et d'analyse des signaux EMG ainsi que la complexité des méthodes de classification des mouvements fins de la main. De même, les défis de développement d'une prothèse mécatronique de main, ayant une taille réaliste, seront abordés. Finalement, un prototype sera conçu. Celui-ci contiendra un ensemble d'électrodes adaptées à l'enregistrement de signaux EMG, une carte d'acquisition et de traitement personnalisée, ainsi qu'un code de prédiction de 13 mouvements de la main, embarqué sur Raspberry Pi. La prédiction sera faite en temps réel avec des résultats d'efficacité allant jusqu'à 95,8% et en moyenne de 92,7%. Enfin, un prototype de prothèse de main mécatronique sera développé.

1.1 Contributions

Ce projet s'est basé sur les constatations rendues par l'état de l'art, qui soulignent la nécessité d'établir un contrôle plus intuitif pour les prothèses myoélectriques. Cela passe d'abord par l'utilisation des différents muscles présents dans l'avant-bras pour une amputation sous le coude, les mouvements du poignet n'étant pas pris en compte dans notre étude. Il s'agit d'exploiter les contractions associées aux muscles extenseurs et fléchisseurs des doigts de la main pour contrôler la prothèse. Une des difficultés actuelles est de pouvoir effectuer une classification des mouvements en temps réel précise et un bon traitement des signaux EMG.

En résumé, les contributions du prototype développé sont les suivantes. Du point de vue de l'instrumentation, la carte d'acquisition et de traitement des signaux possède de très bonnes propriétés de filtrage et permet une acquisition en temps réel. Elle peut également être utilisée avec différentes plateformes de travail et reste très compacte. Du point de vue de la classification, de très bonnes efficacités de prédiction ont pu être observées et 13 mouvements de la main peuvent être distingués. La prédiction est fluide, intuitive et précise tout en demandant peu de complexité de calcul. Cela a conduit à un code embarqué sous Raspberry Pi qui pourrait également être utilisé sur d'autres plateformes. Enfin, un prototype de prothèse de main mécatronique adapté a été développé.

Un article pour la conférence EMBC 2018 [9] a été soumis afin de contribuer aux avancées technologiques des systèmes biomédicaux, et celui-ci a été accepté pour la conférence.

1.2 Objectifs du projet

Le principal objectif de ce projet a donc été de réaliser un nouveau système de détection de mouvements complexes de la main, tels que la flexion indépendante de chaque doigt ou des saisies variées, afin de fournir un panel de mouvements utiles à l'utilisateur. De plus, les accents ont été portés sur la nécessité d'avoir une prédiction de mouvement fluide et très intuitive pour l'utilisateur. Enfin, nous avons voulu réaliser un système de contrôle embarqué en utilisant des éléments peu coûteux afin de minimiser le coût global de la prothèse. Le but de ce projet est de proposer un système de contrôle embarqué ouvrant de grandes opportunités pour le développement de prothèses myoélectriques légères, peu coûteuses et plus intuitives.

1.3 Organisation du mémoire

Ce projet a été soutenu par le CIRRIS (Centre Interdisciplinaire de Recherche en Réadaptation et Intégration Sociale), du CIUSSS de la Capitale-Nationale. De même que par le programme des aides techniques en appareillage pour la clientèle adulte et aînée en déficience motrice, du CIUSSS de la Capitale-Nationale.

La suite du mémoire sera organisée comme suit : La section 2 rendra compte de l'état de l'art sur les prothèses myoélectriques, la section 3 décrira le projet de façon technique, la

section 4 présentera les électrodes et leurs positions sur l'avant-bras, la section 5 le système d'acquisition et de traitement des signaux EMG. La section 6 décrira l'algorithme de classification sous Matlab, tandis que la section 7 portera sur le développement du code Matlab en C++ sur Raspberry Pi Zero. La section 8 présentera le design de la prothèse mécatronique. Enfin, la section 9 donnera les résultats obtenus avec le système conçu et la section 10 analysera les élargissements possibles du projet.

2 État de l'art sur les prothèses myoélectriques

Dans cette partie, nous examinerons l'état de l'art dans le domaine des prothèses myoélectriques. Ce sujet est très souvent abordé dans les articles scientifiques. Les prothèses mécaniques ont longtemps été les prothèses les plus rependues. Celles-ci commandent l'ouverture et la fermeture de la main (ou la flexion de l'avant-bras et le verrouillage du coude) grâce à un système de câblage relié à l'épaule opposée. Elles sont composées, le plus souvent, d'outils interchangeables tels que des mains, des crochets ou des pinces, et possèdent une forte résistance pour effectuer des travaux physiques. L'inconvénient majeur de ce type de prothèse est qu'elles nécessitent beaucoup d'entraînement et de rééducation. De plus, face au manque de variété des prises, de nouvelles prothèses ont commencé à se développer grâce aux progrès technologiques. Les prothèses myoélectriques, dotées d'une certaine facilité de commande, permettent un plus grand nombre de mouvements tout en étant plus pratiques. En effet, elles n'utilisent pas de câbles, pour produire des mouvements, mais uniquement des électrodes afin d'enregistrer les contractions musculaires des muscles résiduels. De nombreuses recherches portent sur l'étude de ces prothèses qui permettraient de faciliter la vie de nombreux patients amputés. Même si beaucoup de prototypes prometteurs voient le jour, l'étape de la mise sur le marché médical reste compliquée. Les exigences sur la validation médicale de ces prototypes sont nombreuses et sont difficiles à satisfaire. En effet, il est, par exemple, difficile de créer des prothèses myoélectriques adaptées à un ensemble d'utilisateurs amputés du fait des variations de niveau d'amputation, qui peuvent limiter le placement des électrodes. De plus, l'altération des muscles des fléchisseurs/extenseurs des doigts est envisageable, et dépend de chaque usager. Il peut donc être difficile, de tester et d'adapter des prototypes afin qu'ils répondent aux besoins et problématiques variés des personnes amputées. C'est pourquoi, malgré des études prometteuses, les prothèses actuellement commercialisées ont des fonctionnalités proches.

Nous allons nous pencher sur certaines de ces avancées afin d'en dégager les aspects essentiels. Nous commencerons par nous pencher sur les prothèses commerciales existantes, puis nous mettrons en évidence certains problèmes rencontrés pour en dégager les améliorations en développement. Puis nous verrons les méthodes de classification des signaux EMG existantes, pour terminer par l'analyse des prototypes en développement ainsi que des améliorations futures. Ainsi, nous aurons une vision plus globale de l'état de l'art sur les prothèses myoélectriques.

2.1 Les prothèses myoélectriques commerciales

Avant toute chose, cette partie va nous permettre de faire un bilan des prothèses actuellement sur le marché médical. Le fonctionnement des prothèses est globalement semblable d'une prothèse à l'autre. Des électrodes fixées dans l'emboîture de la prothèse sont en contact direct avec la peau. Elles captent les signaux musculaires qui sont envoyés à des systèmes de contrôle d'un ou plusieurs moteurs placés dans la main. Les moteurs fonctionnent grâce à l'énergie de piles ou de batteries et déclenchent les mouvements de la prothèse.

Certaines entreprises se distinguent : Ottobock (Allemagne) et Touch bionics par Össur (Islande) sont les leaders du domaine, basés partout dans le monde. La prothèse Bebionic (Ottobock) ou les prothèses i-limb Quantum/Revolution/Ultra (Touch bionics) peuvent par exemple activer de 14 à 36 mouvements préenregistrés.

Voici, plus en détail, les prothèses myoélectriques les plus populaires [10] :

Figure 1 : Main sensor speed (Ottobock) [4].

Cette prothèse (Figure 1) possède un fonctionnement relativement simple : le pouce se met en opposition avec le majeur et l'index, puis la main s'ouvre et se ferme en fonction de la contraction musculaire de l'usager. Les autres doigts suivent le mouvement de manière passive. Cette prothèse est très rapide d'ouverture et de fermeture avec une vitesse maximum de 300mm/s, soit presque le double de la plupart des prothèses de main. De plus, un capteur de frottement est placé dans son pouce permettant de savoir si un objet glisse à l'aide des frictions exercées sur le capteur. La prothèse se resserre alors automatiquement pour retenir l'objet. Enfin, la vitesse et la force de préhension peuvent s'ajuster automatiquement à l'aide des informations provenant d'une jauge placée entre le pouce et l'index.

Figure 2 : Main Michelangelo (Ottobock) [4].

La main Michelangelo (Figure 2) comprend 3 doigts à commande active et 2 suivant de manière passive (annulaire et auriculaire), avec un poignet souple pour faciliter la flexion/extension. Le pouce peut prendre deux positions qui sont contrôlées myoélectriquement en fonction des contractions musculaires : une position latérale ou en opposition avec le majeur et l'index. Elle possède 7 possibilités différentes de préhension et une force de saisie entre 6 et 7kg. Les différentes possibilités de préhension sont effectuées à l'aide du mouvement du pouce, les autres doigts se positionnant par rapport à celui-ci. Cette prothèse ainsi que la suivante sont compatibles avec un poignet rotatif permettant la flexion, l'extension et une très bonne souplesse.

Figure 3 : Main i-limb Ultra Revolution (Touch Bionics) [5].

La main i-limb (Figure 3) possède ses 5 doigts articulés, chaque doigt intégrant un moteur indépendant à sa base. Les doigts étant tous motorisés, il est possible de les reprogrammer afin d'effectuer certaines activités spécifiques, par exemple : l'index pointé pour taper sur un clavier. La prothèse possède une très bonne préhension des objets et peut porter jusqu'à 90kg, les doigts pouvant supporter 32kg. Le pouce de cette prothèse peut être déplacé manuellement pour le mettre en opposition ou non.

Le contrôle de la main fonctionne sur le principe de déclencheurs : maintien ouvert, co-contraction, double et triple contraction du muscle. Une co-contraction est une contraction simultanée des muscles fléchisseurs et extenseurs et permet notamment de changer de type de mouvement (fermer la main, fermer la main en laissant l'index pointé, etc.). Une triple contraction du muscle fléchisseur pourrait, par exemple, permettre de pointer l'index. Un logiciel spécifique (utilisé pour programmer la prothèse) permet d'associer un des déclencheurs à un mouvement préenregistré. Certaines des prothèses i-limb peuvent être contrôlées par une application sur Smartphone afin de donner accès au contrôle en direct de plusieurs prises de la main [11].

Figure 4 : Système i-limb Digits (Touch Bionics) [5].

Ce système (Figure 4) est la solution myoélectrique aux amputations partielles de main. Il incorpore de 2 à 5 doigts de la main i-limb dans une prothèse de main partielle.

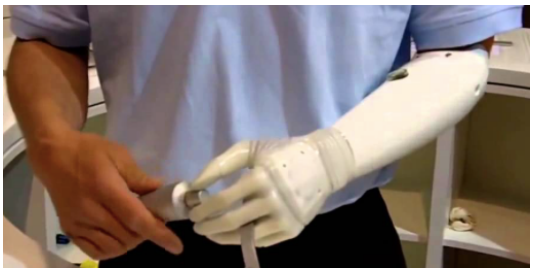

Figure 5 : Main Transcarpienne (Ottobock et Touch Bionics).

Cette prothèse myoélectrique (Figure 5) permet de gérer les amputations longues d'avant-bras ainsi que la désarticulation du poignet. Elle a l'avantage d'être légère et d'avoir une main plutôt courte grâce à la miniaturisation des composants.

Figure 6 : Le greifer (Ottobock) [4].

Le greifer (Figure 6) est une pince-étau myoélectrique avec une force de serrage plus importante qu'une main myoélectrique, idéale pour les travaux manuels et pour manipuler de gros objets. Un ergo orientable à son extrémité permet aussi les prises fines.

Figure 7 : Main bebionic3 (Ottobock) [6].

La prothèse bebionic3 (Figure 7) est considérée comme la plus performante du marché biomédical actuel. Possédant 14 modes de préhension différents basés sur le principe de déclencheurs, elle permet de réaliser les tâches quotidiennes les plus courantes. Tous les doigts sont articulés de façon optimale et permettent une très bonne préhension des objets. Tout comme les prothèses i-limb, le pouce de cette prothèse peut être déplacé manuellement pour le mettre en opposition ou non. Les changements de mode peuvent être activés à l'aide d'un bouton placé sur la prothèse au lieu d'utiliser des co-contractions. Un gant en silicone lui donne la particularité de ressembler à une vraie main humaine. Sa rapidité et sa robustesse contribuent à la rendre populaire sur le marché actuel [6].

Autres prothèses contrôlées myoélectriquement :

(a) **(b)** **(c)**

Figure 8 : **(a)** Main d'enfant, **(b)** poignet Myo Wrist, **(c)** coude Dynamic Arm.

La main d'enfant existe en 4 tailles différentes et est adaptée pour des enfants de 2 à 13 ans. Seuls trois doigts sont actifs, permettant l'ouverture et la fermeture de la main. Le poignet Myo Wrist est une articulation mécanique sur laquelle se fixe une main myoélectrique transcarpienne. Elle maintient bloquée l'articulation du poignet dans différentes angulations, son moteur étant commandé myoélectriquement. Le coude Dynamic Arm réalise le mouvement de flexion/extension de l'avant-bras en verrouillant ou déverrouillant l'articulation selon les impulsions musculaires reçues. Son moteur est intégré dans le coude.

Il existe évidemment d'autres prothèses myoélectriques commerciales, mais nous ne nous étendrons pas plus sur le sujet. Les avantages de ces prothèses commerciales sont l'adaptation facile du système à différents utilisateurs, la robustesse (la prothèse possède un comportement très semblable au cours du temps, à un même type de contraction on pourra attendre la même réponse de la prothèse) et le contrôle simplifié : la contraction d'un seul muscle est nécessaire. Le principal inconvénient étant le manque d'intuitivité dans le contrôle et le nombre limité de degrés de liberté pour certaines prothèses. Il faut retenir que même si les prothèses proposées sont nombreuses, des recherches et développements sur le sujet sont en cours pour améliorer les performances myoélectriques de ces mains robotiques. Le tableau suivant a été réalisé afin de pouvoir comparer la prothèse conçue dans ce projet (Tableau 7) avec les prothèses commerciales existantes.

Tableau 1 : Comparaisons des prothèses.

Prothèses	**Prix approximatifs**	**Caractéristiques principales**
Main sensor speed (Ottobock)	9 000 $USD	- Pouce contrôlé activement, autres doigts suivent passivement - Très rapide d'ouverture et de fermeture - Capteur de frottement pour prévenir le glissement
Main Michelangelo (Ottobock)	40 000 $USD	- 3 doigts contrôlés activement, autres doigts suivent passivement - 7 possibilités de préhension

		- Très bonne adaptation lors de la saisie d'objets
Main i-limb Ultra Revolution (Touch Bionics)	33 000 $USD	- 5 doigts contrôlés activement - Possibilité d'une programmation spécifique de chaque doigt - Très bonne préhension et port de charges lourdes
Système i-limb Digits (Touch Bionics)	35 000 $USD pour 5 doigts	- Pour amputations de main partielles : 2 à 5 doigts - Utilise les doigts de i-limb
Main bebionic3 (Ottobock)	25 000-35 000 $USD	- 14 modes de préhensions différents - 5 doigts contrôlés activement - Très bonne préhension - Rapide et robuste

2.2 Les problèmes rencontrés

Nous allons voir que beaucoup de chercheurs se penchent actuellement sur les améliorations à apporter aux prothèses myoélectriques existantes. Car même si celles-ci semblent déjà très développées, les avancées technologiques permettent de pousser les performances encore plus loin. L'étape la plus compliquée restant la validation du fonctionnement de la prothèse et les tests sur des patients amputés avant commercialisation.

Plusieurs difficultés se posent lorsqu'il s'agit des prothèses myoélectriques. Tout d'abord, la difficulté pour l'usager de contrôler facilement la prothèse. En effet, les signaux myoélectriques sont particulièrement faibles en amplitude (entre -5mV et 5mV) et difficiles à produire (maintien du signal dans le temps) : sans un algorithme de contrôle des moteurs et un bon traitement des signaux, il est compliqué d'utiliser la prothèse.

Aujourd'hui, beaucoup de prothèses ne contrôlent que quelques degrés de liberté et enregistrent des mouvements prédéfinis que l'on peut reproduire en contractant un seul muscle 1 à 3 fois de suite. L'avantage de cette technique étant la facilité pour l'utilisateur de produire des contractions, car un seul muscle est réellement sollicité. De plus, la prothèse est moins sujette aux erreurs d'interprétations de mouvements et donc plus robuste. L'inconvénient majeur de ces prothèses étant le manque d'intuitivité. L'utilisateur doit se focaliser sur le nombre de contractions à faire pour produire un mouvement particulier. Ainsi, la difficulté du contrôle myoélectrique de la prothèse, de par l'analyse délicate des signaux myoélectriques, est un des principaux problèmes rencontrés.

La seconde difficulté rencontrée est le manque de retour de sensation fourni à l'usager. Les ordres de mouvements sont envoyés à la prothèse, mais l'utilisateur ne reçoit, le plus souvent, pas d'informations sur l'état de la prothèse. Cela crée un manque d'intuitivité dans le contrôle et parfois une incompatibilité entre la prothèse et son utilisateur. C'est pourquoi généralement, les prothèses choisies par les usagers sont des prothèses classiques : un crochet par exemple sans contrôle myoélectrique. Ainsi, l'utilisateur perçoit directement les actions qu'il effectue. Car avec des prothèses myoélectriques, l'usager n'a pas d'information sur le signal de contrôle ou sur la position finale de la prothèse. Il est forcé d'écouter les moteurs et de voir combien de temps le système met pour changer de position : or cela ne remplacera jamais la perception des signaux nerveux.

Enfin, une dernière difficulté est que, pour beaucoup de patients, la prothèse permet de restaurer leur image et donc ils souhaitent une prothèse esthétique. Ce qui ne permet pas toujours de pouvoir développer la maniabilité. De plus, le prix ne devant pas être trop élevé, des concessions doivent être faites.

2.3 Les améliorations en développement

Face à ces problèmes de manque d'intuitivité, de retour de sensation, de maniabilité, et de coût élevé, des améliorations sont en cours de développement afin de permettre une meilleure utilisation des prothèses myoélectriques. Nous allons voir dans cette partie les domaines porteurs.

2.3.1 Électrodes implantées

Remplacer les électrodes de surface par des électrodes implantées fait partie des recherches actuelles sur les prothèses, [12], [13]. En effet, il faut savoir que, plus la distance entre les électrodes et les muscles est grande, moins le signal sera correctement enregistré. Le contrôle de la prothèse dépendant de la qualité des signaux électriques capturés, des chercheurs travaillent sur l'ajout d'électrodes implantées sur chaque muscle du bras. Néanmoins, même avec cette méthode, il reste difficile de reconnaître exactement les potentiels d'actions associés à des mouvements. Dans un bras valide, les muscles travaillent ensemble et il faudra donc comprendre ce fonctionnement afin de pouvoir contrôler chaque mouvement de la main.

2.3.2 Les capteurs de force sensible

Dans certains prototypes, des capteurs de force (FSR : Force Sensitive Resistor) sont ajoutés afin d'obtenir des informations sur l'effort de serrage de la prothèse. Normalement, ces capteurs fournissent un signal proportionnel à la force appliquée, dépendamment de la résistance du capteur. Certains circuits, même très petits, permettent de donner un signal linéaire et proportionnel comme l'explique l'article [14]. Mais le problème, présent ici, réside dans le fait que les utilisateurs ne parviennent pas à exercer une force suffisante pour que le capteur la perçoive. Les capteurs FSR pourraient être recouverts d'un tampon conique élastique d'environ 8mm d'épaisseur pour plus de sensibilité, [14].

2.3.3 Le retour de sensation

Le manque de retour de sensation est un des problèmes majeurs dans les prothèses myoélectriques actuelles. Le principe de retour de sensation est détaillé dans l'article [8] et présenté à la figure 9.

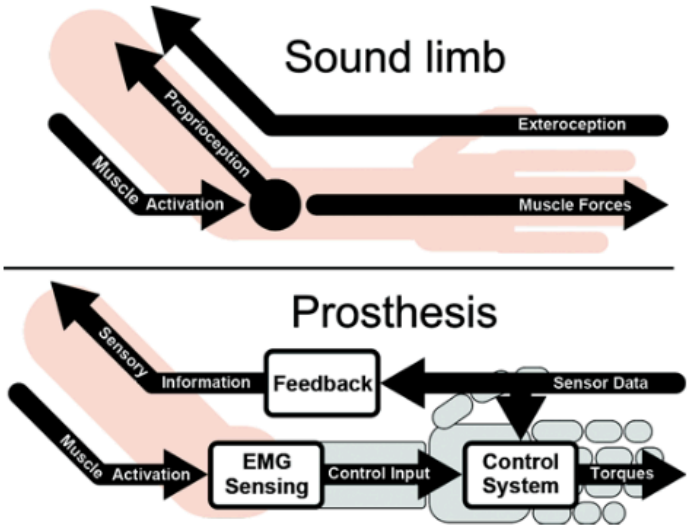

Figure 9 : Comparaison du signal qui traverse l'avant-bras humain et de celui provenant des prothèses myoélectriques modernes. Image provenant de l'article [8].

Le signal qui circule entre le bras de l'amputé et la prothèse peut être divisé en plusieurs parties : l'intention de l'utilisateur, le contrôle du mouvement et le retour d'information provenant des capteurs. Le système de contrôle permet de contrôler la prothèse et un système de retour d'information fournit à l'utilisateur des sensations artificielles à l'aide de capteurs. Cela rend le contrôle de la prothèse non invasif pour le patient. De nombreuses recherches actuelles, se basant sur ce principe, tentent de développer des méthodes pour fournir ce genre de sensation à l'usager [15], [16] et [17].

Il a été dégagé que plusieurs paramètres seraient utiles à renvoyer à l'utilisateur. La force de préhension est un élément essentiel à retourner. Il est, notamment, très important d'appliquer la bonne force dans le cas d'objets fragiles ou d'interaction avec des êtres vivants. La position des doigts peut, également, être un paramètre à retourner. Enfin, des indications sur l'état général du système de contrôle sont nécessaires pour aider l'utilisateur à appréhender sa prothèse. Il pourrait, par exemple, savoir si un mouvement de préhension est fini ou non.

Il faut maintenant se demander si le retour d'information doit être continu ou discret. Un mode continu peut permettre le contrôle intuitif de la prothèse assez facilement, mais la perception de signaux non physiologiques par l'utilisateur pourrait le perturber. Les signaux discrets pourraient avoir l'avantage de moins perturber l'utilisateur, ils donnent des informations sur l'état du système de façon ponctuelle, mais sont moins précis que les signaux continus [18]. Les informations envoyées devront également être intuitives et relativement simples.

Pourtant, malgré les avantages certains du retour de sensations [19], sa mise en place reste délicate. En effet, la méthode employée doit être non intrusive et confortable pour l'utilisateur. De plus, il est nécessaire d'ajuster le retour d'information à chaque usager. Des études se sont donc penchées sur la mise en place de méthodes et technologies pour renvoyer

des informations utiles à l'utilisateur [20]. Une première méthode invasive consisterait à stimuler directement les nerfs concernés [21], mais elle est encore à l'état expérimental. Une seconde se base sur la mesure de la force par un capteur, qui sera retransmise à l'utilisateur en appliquant une force similaire sur la peau. La stimulation électrotactile [22] est la troisième solution envisagée. La force est alors modulée en amplitude électrique et appliquée sur les endroits de la peau concernés. Le problème majeur de cette méthode étant qu'une douleur est généralement ressentie par l'utilisateur probablement à cause des différents types de mécanorécepteurs qui sont activés simultanément. La dernière solution envisagée est la stimulation vibrotactile [23] qui est moins douloureuse que la stimulation électrotactile. De même que pour la stimulation précédente, l'intensité du stimulus perçue est fortement liée à l'intensité du stimulus appliquée. Il en a été déduit que cette méthode semblait confortable pour l'utilisateur, mais que son efficacité restait limitée. En effet, il n'a pas pu être constaté une grande différence sur les performances d'utilisation de la prothèse avec et sans retour d'informations.

De plus, une adaptation de l'usager aux stimuli est possible [24]. Ce problème dépend évidemment de la sensibilité de l'utilisateur ainsi que du placement des électrodes. S'il survient, l'intensité du stimulus doit être ajustée. Pour réduire ce phénomène d'adaptation, des stimuli par intermittence peuvent être envoyés ou un ajustement de fréquence peut être réalisé. Dans le cas de la stimulation électrotactile : l'adaptation est plus faible pour des courants élevés de stimulation (en dessous du seuil de douleur).

Les résultats de ces recherches ont permis de démontrer que le retour de position aidait l'utilisateur à mieux se rendre compte de la taille de l'objet saisi. De plus, cela peut favoriser le phénomène de membre fantôme et donc permettre à l'utilisateur de générer des signaux myoélectriques plus importants. La stimulation vibrotactile semble être une méthode prometteuse par rapport à la stimulation électrotactile qui est douloureuse pour le patient. Après quatre semaines d'utilisation de la stimulation vibrotactile, aucun effet indésirable sur la peau n'a pu être constaté. Il faudra par la suite travailler sur la taille des électrodes de stimulation ainsi que sur leur nombre. En effet, une électrode ne peut être utilisée que pour transmettre un seul paramètre (comme la position ou la force) et sa position est très importante pour faciliter la compréhension des informations par l'usager.

2.3.4 Maintien automatisé des objets

Dans un cas idéal, l'utilisateur doit pouvoir contrôler la saisie, le poignet, et initier le mouvement de saisie. La prothèse peut alors éventuellement accomplir la fin du mouvement de façon automatique et continuer de tenir l'objet pour permettre à l'utilisateur de se concentrer sur le déplacement de l'objet avec le bras et le poignet. Dans des prototypes actuels et dans des prothèses déjà commercialisées, par exemple la Main Sensor Speed (Ottobock, voir plus haut) [4], la prévention du glissement de l'objet est assurée [25]. Des études portent sur les différents mouvements de préhension existant afin d'adapter les prises de la prothèse à l'objet, comme le montre la Figure 10.

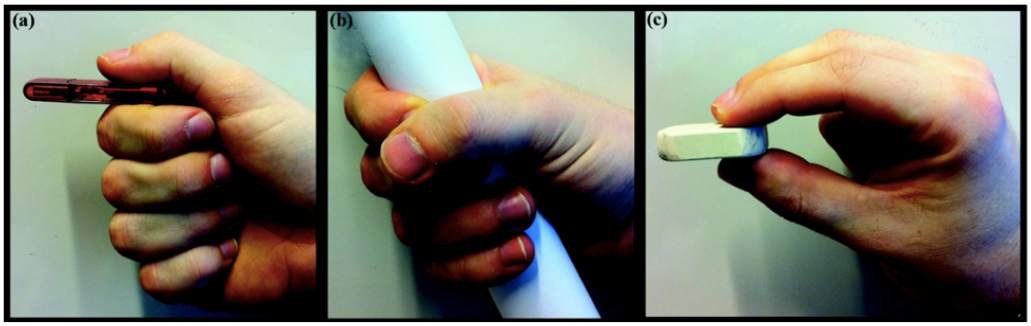

Figure 10 : Modes de préhension principaux de la main : latéral **(a)**, cylindrique **(b)** et tripode **(c)**. Image provenant de l'article [8].

2.3.5 Contrôle de la force et de la vitesse

La force et la vitesse sont des paramètres liés d'un point de vue mécanique : ils sont proportionnels. Il faut donc, la plupart du temps faire un choix entre contrôler l'un ou l'autre. Dans la majorité des prothèses actuelles ainsi que pour les prototypes, c'est la force qui est choisie. L'amplitude des signaux EMG étant proportionnelle à la force désirée, l'ajustement de celle-ci peut se faire par une simple mesure d'amplitude. Deux modes de contrôle de la force peuvent être utilisés :

- Un contrôle direct de la force par l'utilisateur.
- Appliquer automatiquement une force suffisante pour maintenir l'objet tout en donnant la possibilité à l'utilisateur de contrôler la force durant la saisie.

Le choix entre ces deux modes est déterminé par les attentes des utilisateurs ou par le cahier des charges retenu. Dans certains prototypes, la vitesse peut être choisie comme élément de contrôle au lieu de la force (par exemple, le prototype FluidHand [26] détaillé plus bas), car elle favorise la fluidité de mouvement.

2.3.6 Temps d'exécution de la prothèse

Le temps d'exécution de la prothèse est un élément primordial pour un fonctionnement efficace. Il faut s'assurer que l'utilisateur ne ressente pas de délai trop important entre la commande d'un mouvement de la prothèse et sa réponse. Pour cela, la fenêtre d'observation des signaux EMG est importante. Un compromis doit être fait, car une petite fenêtre d'observation réduit le délai, mais peut induire des erreurs dans l'interprétation du mouvement désiré par l'utilisateur. Il faut, de plus, tenir compte du temps de calcul des informations reçues. Ce schéma (Figure 11) résume assez bien le travail qui doit être effectué entre l'ordre envoyé par l'utilisateur et la réponse de sortie du système de contrôle.

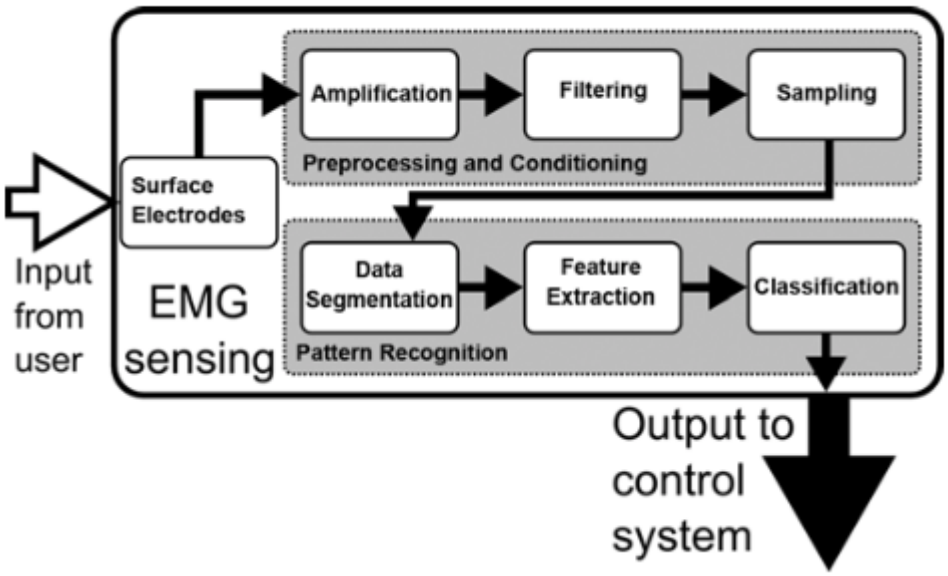

Figure 11 : Résumé du chemin entre l'entrée et la sortie du système de la prothèse. Image provenant de l'article [8].

Des études ont montré qu'une fenêtre devait être analysée sur moins de 300ms [27], sinon le délai devenait perturbant. Plus récemment, ce temps se trouve entre 100 et 120ms, mais la précision sur le contrôle de la prothèse diminue. Un autre facteur important est à prendre en compte : le nombre de signaux utilisés pour activer un certain mouvement. Plus ce nombre est important, plus les délais de traitement sont longs. Enfin, les délais dépendent également des actionneurs utilisés : dans la plupart des mains myoélectriques, des moteurs à courant continu (moteur DC) sont employés. Ces moteurs ne sont pas forcément les plus rapides, car pour fournir des couples nécessaires au fonctionnement d'une prothèse, la vitesse du moteur en est impactée.

2.3.7 Contrôle simultané du poignet et de la main

Actuellement, des poignets fonctionnent myoélectriquement, mais de manière dissociée des mouvements de la main. Pourtant, le poignet et la main bougent souvent simultanément, il faudrait donc travailler sur un contrôle simultané des deux (en utilisant le même signal myoélectrique par exemple). L'établissement d'un contrôle proportionnel pourrait également être une option. Des études portent sur ce problème afin de permettre deux mouvements de liberté du poignet en association avec les mouvements de main [28], [29].

2.3.8 Motorisation du coude à l'aide de câbles de position

Alors qu'aujourd'hui des coudes sont contrôlés à l'aide de signaux myoélectriques pour la flexion/extension, certains utilisateurs préfèrent se pencher sur une manipulation plus intuitive du coude. Un dispositif, composé d'un transducteur linéaire actionné par un câble de position et d'un potentiomètre linéaire, permet, en fonction de la tension dans le câble,

d'ajuster la position du coude tout en gérant la force à appliquer. Ce système, très pratique, sépare le contrôle de la main et du coude et permet de stopper le circuit quand le coude est en position de repos. De plus, utiliser des câbles pour le coude permet un retour aux choses plus intuitives pour l'utilisateur.

Il existe également des études mettant en œuvre un système de contrepoids qui permet de maintenir le coude en équerre à 90°. Le système de commande joue sur la tension du câble : plus il y a de tension, plus le coude se fléchit et inversement pour l'extension. Très peu de force est alors nécessaire à l'utilisateur pour fléchir le coude lorsqu'il compense le poids de l'avant-bras et de la prothèse. Quand une charge plus importante est appliquée à l'extrémité de l'avant-bras, il faut plus de force pour provoquer un mouvement, ce qui donne à l'utilisateur une idée de la charge et lui fournit une meilleure perception. On parle ici de favoriser la proprioception de l'utilisateur, c'est-à-dire de développer sa perception psychologique [14].

2.3.9 Utilisation du mouvement de l'épaule pour contrôler la prothèse

L'épaule pourrait fournir deux signaux de contrôle indépendants selon deux mouvements de l'épaule bien distincts (élévation/dépression de l'épaule et protraction/rétraction). Cela donnerait un bon contrôle de deux mouvements de liberté de la prothèse, car les signaux utilisés ont l'avantage d'être forts et distinguables. Il faut, par contre, trouver une interface appropriée pour collecter les signaux myoélectriques. Les détails de ces recherches peuvent être trouvés dans l'article [30].

2.3.10 Utilisation de l'humérus

Lors de recherches, on a pu se rendre compte que beaucoup d'amputés savaient bouger leurs humérus, mais que la prothèse ne suivait pas le mouvement. L'article [14] détaille différents objectifs de certaines recherches. Ainsi, un des objectifs a été de permettre une rotation externe de la prothèse grâce à la rotation interne de l'humérus. Des méthodes se sont développées pour réaliser cela. Il est notamment possible d'attacher la prothèse à l'humérus par une opération chirurgicale : c'est faire une ostéointégration [31]. Malheureusement, cela reste peu exploité, car la nécessité d'une chirurgie et de rééducation ne motive pas les patients, et cette opération n'est pas possible pour tous.

Une autre méthode consiste à insérer un implant en titane en forme de T dans l'extrémité de l'humérus afin de retransmettre le couple et donner le contrôle de la rotation interne. Beaucoup d'autres capteurs permettent de gérer ce contrôle intuitif de la rotation : tourne-disque de friction, rotateur huméral ou même un aimant transversal pour que la rotation puisse être ressentie directement.

Comme nous avons pu le voir, afin d'ajouter des fonctionnalités aux prothèses existantes, beaucoup de pistes différentes peuvent être explorées, allant de l'emploi d'électrodes implantées à l'amélioration du retour de sensation en passant par l'utilisation d'autres parties du corps comme l'épaule ou l'humérus. Ainsi, nous pouvons constater que des améliorations sur les prothèses sont en cours de développement afin de pallier à des

problèmes rencontrés lors de leurs utilisations. Mais cela permettrait également de renforcer la cohésion entre le patient amputé et sa main myoélectrique.

2.4 Les méthodes de classification des signaux

Les signaux provenant des muscles sont difficiles à analyser, c'est pourquoi le plus souvent, les prothèses myoélectriques sont limitées à un degré de liberté. Ainsi, des modèles mathématiques sont mis en place pour distinguer différents mouvements avec précision et donc pouvoir augmenter le nombre de degrés de liberté possible pour la prothèse. Les algorithmes permettent d'analyser les signaux EMG sous différents angles, en utilisant des paramètres temporels ou fréquentiels de ces signaux.

Il existe de nombreuses méthodes de classification telles que : Linear Discriminant Analysis (LDA), Support Vector Machine (SVM), k-Nearest Neighbor (kNN), etc. Différentes méthodes de classification et d'extraction de paramètres discriminants, pour les signaux EMG, sont analysées dans [32], [33] et [34]. Ces études montrent que choisir des paramètres bien adaptés à une méthode de classification donnée permet d'augmenter considérablement la précision de classification. Beaucoup de classifieurs sont comparés en matière d'efficacité dans l'article [34].

Les auteurs de l'article [33] ont démontré que les résultats de classification dépendent du type de paramètre sélectionné et ils ont également mis en évidence la redondance de certains paramètres dans le domaine temporel et fréquentiel. Dans cet article, 37 paramètres discriminants sont analysés et le maximum d'efficacité de discrimination est atteint par le modèle Autorégressif (AR) d'ordre 4 avec plus de 92,1% de réussite. Ces résultats démontrent la nécessité de choisir le bon classifieur avec un ensemble de paramètres adaptés pour avoir un bon pouvoir discriminant des signaux EMG.

Nous pouvons dissocier les méthodes qui n'utilisent aucun apprentissage préalable, et les méthodes de classification, comparables à des machines d'apprentissages. Ici, certaines méthodes seront analysées et leurs performances pourront être comparées.

2.4.1 Méthode sans apprentissage préalable

- *Méthode des prothèses commerciales actuelles (SOA: State Of the Art) :*

  Cette méthode regroupe des commandes classiques mises en œuvre dans les prothèses commerciales. Les usagers adaptent leurs prothèses à leurs propres utilisations dans la vie de tous les jours, grâce à une interface de contrôle. Dans les prothèses myoélectriques les plus courantes, deux canaux d'électrodes (2 paires d'électrodes) sont utilisés (un sur le fléchisseur et un sur l'extenseur de l'avant-bras). Cela permet alors le contrôle direct d'un degré de liberté (deux mouvements) grâce à l'analyse de l'amplitude des signaux EMG. Afin de changer de degré de liberté, un signal de commutation doit être généré, provoquant le changement d'état de la prothèse. Cette méthode est souvent étendue afin de disposer de presque 4 degrés de

liberté. L'utilisateur peut alors choisir quatre mouvements spécifiques qu'il souhaite faire à la prothèse, et sélectionne ces mouvements par 1, 2 ou 3 co-contractions de son muscle.

- *Méthode Teager-Kaiser Energy (TKE) :*

Cette méthode se base sur un opérateur d'énergie pouvant être utilisé sur des signaux EMG de surface. L'article [35] présente cet opérateur et ses fonctionnalités. Il permet de détecter l'activité d'un muscle et d'amplifier le signal utile de l'EMG tout en évitant de traiter du bruit. La formule couramment utilisée pour trouver la forme TKE d'un signal est la suivante :

$$\mathrm{TKE}(x_n) = x_n^2 - x_{n-1}.\, x_{n+1} \qquad (1)$$

Où, x_n est un échantillon à un instant donné, x_{n-1} et x_{n+1} sont les échantillons à l'instant précédant et suivant respectivement. x_n représente le plus souvent le signal EMG provenant des électrodes. Cet opérateur mesure les changements instantanés d'énergie du signal et reste relativement simple en comparaison d'autres méthodes de traitement de signal.

Cette méthode TKE est, le plus souvent, utilisée pour détecter le début des mouvements à l'aide de seuils d'amplitude. Son calcul s'effectue en temps réel, même si un léger retard, dépendant de la fréquence d'échantillonnage, sera toujours observé dans le calcul du TKE, car il faut pouvoir avoir accès à l'échantillon à l'instant t+1 pour le calcul du TKE à l'instant t. Après application de l'opérateur au signal EMG, celui-ci est amplifié en énergie, le bruit devenant négligeable (optimisation du SNR (Signal to Noise Ratio)). Ainsi, il est alors plus facile de mettre en place des seuils précis qui seront dépassés lors d'une réelle activité du muscle. L'utilisation de la forme TKE d'un signal EMG permet, en outre, de diminuer les erreurs d'interprétation sur les mouvements. Ainsi, la méthode TKE pourrait être incorporée à des algorithmes de détection des signaux EMG afin d'en augmenter la robustesse et permettre une meilleure détermination des mouvements.

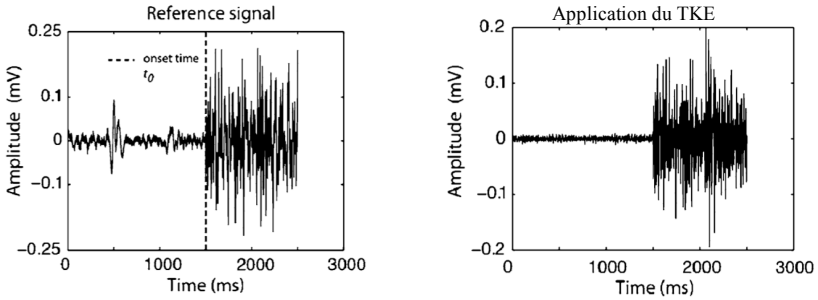

Figure 12 : Signal brut et signal après application de la méthode TKE. Image provenant de l'article [27].

2.4.2 L'intelligence artificielle

- *Méthode LDA :*

Cette méthode est une référence dans la reconnaissance des signaux EMG, c'est la plus commune méthode d'apprentissage pour le contrôle des prothèses. Elle pourrait être définie comme un moyen de reconnaître des mouvements par analyse discriminante linéaire. Le détail du fonctionnement de la méthode LDA peut être trouvé dans [36]. Des paramètres tels que la variance, l'espérance ou la moyenne sont calculés pour différents mouvements. Suite à cela, si les paramètres associés à un mouvement en particulier sont reconnus, alors la prothèse effectue le mouvement en question. Cette méthode est considérée comme une technique rapide, de bonne efficacité et robuste au fil du temps. L'article [37] présente l'utilisation de la méthode LDA dans le cas de la reconnaissance de 8 mouvements de la main en utilisant seulement deux électrodes pour la mesure des signaux EMG. On peut notamment y trouver une discussion sur les meilleurs paramètres discriminants à utiliser afin d'optimiser l'efficacité de prédiction.

- *Méthode Common Spatial Patterns (CSP) :*

Cette méthode se base sur la reconnaissance de modèles spatiaux communs. L'article [38] explique en détail cet algorithme de contrôle. Un estimateur proportionnel a été développé afin d'être capable de cartographier, à grande vitesse, les signaux EMG sur la surface des muscles à l'aide de méthodes de filtrage. Cet algorithme s'appuie sur les combinaisons linéaires spatiales liées à l'amplitude des signaux EMG. Les coefficients sont tirés de filtres d'optimisation de modèles spatiaux qui permettent de mieux distinguer les mouvements voulus. La méthode CSP est utilisée comme un filtre sur des signaux bruts EMG.

De plus, il existe, dans cette méthode, un facteur de contraste qui permet, lorsque le mouvement d'entrée n'est pas bien reconnu, de diminuer le contraste entre les classes (mouvements à reconnaître) et d'atténuer la valeur de sortie (souvent la vitesse). Ainsi, la prothèse n'effectue pas de mouvement plutôt que de se tromper. La CSP se base sur des propriétés mathématiques de séparation de mouvements, de sorte que même les mouvements subtils peuvent être repérés.

Ces méthodes d'apprentissage permettent de rendre les prothèses plus intuitives pour l'usager en comparaison des méthodes sans apprentissages préalables.

2.4.3 Comparaison de certaines méthodes par l'expérimentation

L'article [38] a mis en place une séance de test avec des patients amputés déjà entraînés à développer leurs signaux EMG, et d'autres patients non amputés. Le but étant de bouger leurs prothèses sur 4 états différents :

- Supination / Pronation
- Flexion / Extension
- Main ouverte / Main fermée latéralement
- Main ouverte / Main fermée de côté (tripode grip)

Une série de mouvements à réaliser a été déterminée afin de juger de la maîtrise de 1 à 3,5 degrés de liberté (0,5 car la main ouverte revient deux fois comme état opposé de la main fermée). Ainsi, les performances des méthodes LDA, SOA et CSP ont pu être comparées.

Il en a été déduit que, pour les mouvements complexes où plusieurs degrés de liberté doivent être maitrisés, la méthode CSP était plus performante et rapide que les autres. Pour les mouvements simples, la CSP est moins rapide que la LDA, elle-même moins rapide que la SOA. En effet, la méthode courante (SOA) est beaucoup plus avantageuse lorsqu'on doit travailler avec des mouvements simples qui ne nécessitent pas de changer de type de degré de liberté. Par contre, elle est très peu pratique pour effectuer des mouvements complexes, car elle manque d'intuitivité : il faut se souvenir des mouvements enregistrés et du nombre de contractions à effectuer pour y parvenir. L'algorithme CSP a dépassé les performances de la méthode LDA pour les mouvements complexes en démontrant sa sécurité d'utilisation et sa robustesse. Chaque sujet est parvenu à effectuer les mouvements voulus avec la prothèse sans trop se fatiguer et la plupart ont trouvé que la prothèse été plus contrôlable et naturelle d'utilisation avec la méthode CSP ou LDA.

On peut donc en conclure, selon cette étude, qu'on ne devrait pas étendre le contrôle d'un degré de liberté à plusieurs comme le font les prothèses actuelles. Mais il faudrait développer d'autres méthodes : comme les méthodes d'apprentissages qui possèdent un fort potentiel.

Il existe encore bien d'autres méthodes de classification ou d'algorithmes actuellement en cours d'étude. Par exemple, utiliser la transformée en ondelette (WPT : Wavelet Packet Transform) pour la reconnaissance des formes EMG en temps réel [39]. Ou, proposer d'étendre la classification des signaux pour des mouvements simultanés [40] (car pour l'instant les mouvements sont classifiés de façon séquentielle). Des articles présentent des expériences faites avec de nombreuses électrodes ou de plus en plus d'estimateurs (article [41]), afin de : déterminer les meilleurs paramètres pour la classification, réduire les canaux d'entrées et donc la redondance, et rendre la classification robuste. Toutes ces recherches pourront être très prometteuses pour les prothèses futures.

2.5 Les prototypes en développement

Dans cette partie nous nous pencherons sur les différents prototypes en développement et leurs innovations technologiques. Nous verrons que beaucoup de prototypes semblent posséder des fonctionnalités bien supérieures aux prothèses commerciales, pourtant ils ne parviennent pas pour le moment à percer sur le marché biomédical très strict. Voici un certain nombre de prototypes en développement comparés dans l'article [8] :

- *MANUS Hand* [42] *:*

  Cette prothèse permet un contrôle direct de la force par l'utilisateur. Elle possède un système permettant de maintenir un objet saisi à moins d'avoir reçu un

ordre contraire. Le retour de sensation envoyé à l'utilisateur est assuré par une stimulation vibrotactile.

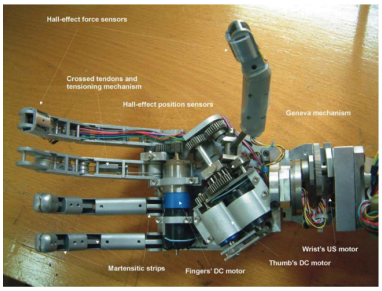

Figure 13 : Prothèse MANUS Hand.

- *AR III Hand* [43] *:*

Ce prototype permet le contrôle de tous les doigts pour la saisie d'objet, mais ne possède pas de système de contrôle de force ou de vitesse. La méthode d'analyse des signaux EMG se base sur du traitement en temps réel.

- *FluidHand* [44] *:*

Cette prothèse permet à l'utilisateur de contrôler la vitesse. De plus, comme pour MANUS Hand, un système permettant de maintenir un objet saisi est en place. Le retour de sensation fourni à l'utilisateur est assuré par une stimulation vibrotactile. Une classification des signaux EMG est également mise en place.

L'avantage majeur de ce prototype est la fluidité des mouvements et une bonne préhension des objets grâce aux cinq doigts articulés de la main. Tous les actionneurs sont pneumatiques et permettent une force de préhension importante. Un actionneur pneumatique transforme une pression en rotation et possède un bon rapport poids/puissance ainsi qu'une très bonne stabilité en haute pression. Son inconvénient est que les recharges des actionneurs sont volumineuses (énergie hydraulique) et compliquées à placer. La miniaturisation de ces recharges est en cours d'étude. Pour contrôler la pompe et les vannes, un microcontrôleur est nécessaire. Un capteur de force résistif se trouve dans l'index et le pouce. Du point de vue du contrôle, la prothèse effectue cinq formes de préhension différentes en fonction des tâches de l'usager. L'utilisateur enregistre différents types de saisies et un système de contrôle permet de distinguer les différents motifs enregistrés pour donner l'ordre de mouvement à la prothèse. L'article [26] présente les performances du prototype FluidHand MK III.

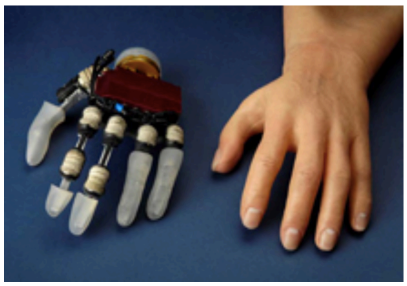

Figure 14 : Prothèse FluidHand.

- *Southampton Hand* [45] *:*

Cette main robotique possède un contrôle direct de la force par l'utilisateur, elle tient automatiquement les objets et empêche le glissement, qui est détecté par des capteurs acoustiques. Southampton Hand applique automatiquement une force suffisante pour tenir un objet, mais donne la possibilité de contrôler la force durant la saisie. La reconnaissance des mouvements est faite par classification EMG.

Figure 15 : Prothèse Southampton Hand.

- *CyberHand* [46] *et* [47] *:*

Cette main robotique permet le contrôle de tous les doigts pour la saisie d'objet, et l'utilisateur peut avoir accès à un contrôle direct de la force. La référence [48] présente le projet CyberHand. La prothèse contient des capteurs de force nécessaires pour prévenir le glissement d'objet, mais ne possède pas un temps de réponse assez court pour l'utiliser. Elle apporte, en plus, le retour de sensation à l'utilisateur grâce à la stimulation vibrotactile. La méthode d'analyse des signaux EMG se base sur un traitement en temps réel.

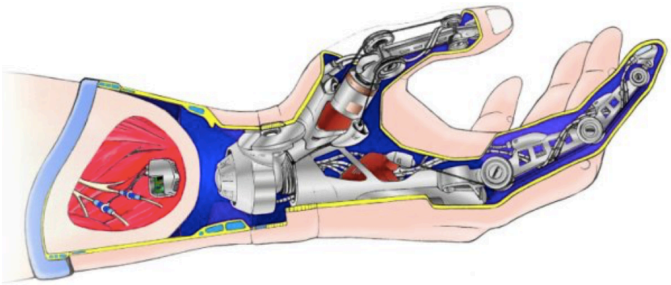

Figure 16 : Prothèse CyberHand.

Le contrôle de tous les doigts est une opération délicate pour l'utilisateur, car il est encore difficile de parvenir à contrôler et maintenir tous les doigts dans la position voulue. Cependant, les prototypes présentés possèdent de très bonnes fonctionnalités qui promettent encore plus d'innovations technologiques pour l'avenir.

2.6 Améliorations futures

Dans cette partie, nous allons détailler des améliorations possibles pour les prothèses myoélectriques de demain. La plupart des méthodes présentées sont encore au stade expérimental.

2.6.1 Avoir de meilleurs protocoles expérimentaux et analyser les attentes des usagers

Un problème majeur récurrent dans beaucoup d'études sur les prothèses, et de parvenir à faire valider des prototypes ou des études. Cela passe d'abord par de bons protocoles de recherches : il faudrait dans l'ensemble améliorer la qualité de la recherche. En effet, certains chercheurs n'utilisent pas beaucoup de sujets pour les tests. Ce raisonnement pourrait mener à des erreurs, car la manière de produire des signaux myoélectriques diffère d'une personne à l'autre. Intégrer plus de participants, et notamment des personnes amputées, dans les études semble être une bonne approche pour obtenir des résultats valides [14].

De plus, les chercheurs ont tendance à collecter des informations en une session très longue sans laisser de temps de repos aux utilisateurs. L'adaptation à une nouvelle prothèse demande du temps et de l'énergie qu'il faut prendre en compte dans les études, pour les personnes non expérimentées avec la prothèse. Enfin, les protocoles et la manière d'acquérir les données devraient être revus, car les protocoles sont les mêmes pour tous, or beaucoup d'usagers (notamment des personnes amputées) fonctionnent différemment vis-à-vis des prothèses. L'intégration de thérapeutes dans certaines études pourrait être une solution, car leurs connaissances des patients seraient un atout majeur.

Les auteurs des articles [8] et [49] souhaitent comprendre les attentes des utilisateurs. Dans l'atelier mis en œuvre dans l'article [8], les participants sont : des techniciens, des cliniciens, des personnes qui travaillent avec des amputés utilisant déjà des prothèses, 9 hommes et 10 femmes représentatifs (thérapeutes, physiciens de réhabilitation, prothésistes…) et d'ingénieurs. Dans cette étude, cinq activités de la vie courante, divisées en plusieurs tâches (le retour d'information, le contrôle de la prothèse, les mouvements du poignet et de la main), seront analysées afin d'en dégager des requis nécessaires pour les prothèses futures. Ce genre d'étude permet alors de dégager des aspects essentiels sur l'utilisation des prothèses myoélectriques. Voici notamment, le tableau des exigences retenues grâce à l'atelier, pouvant être attendues par une personne amputée.

Tableau 2 : Exigences pouvant être attendues par une personne amputée. Tableau provenant de l'article [8].

Subsystem	Number	Requirement
EMG Sensing	1	Multiple wrist movements and grasp types should be easily selectable.
	2	Time delay should be short enough to not disturb user.
	3	User should be able to indicate desired speed of wrist movements and force of grasps.
	4	Wrist movement and grasp type should be simultaneously distinguishable.
Control	1	Available grasp types: cylindrical grasp, tripod grasp, lateral grasp.
	2	Available wrist movements: flexion/extension and rotation.
	3	Prosthesis should automatically continue holding an object once grasped.
	4	Prosthesis should automatically prevent slipping of any held objects.
	5	Grasp execution time should not disturb user.
	6	User should be able to directly control speed of wrist movements and force of grasps.
Feedback	1	Continuous and proportional feedback on grasping force should be provided.
	2	Position feedback should be provided to user.
	3	Interpretation of stimulation used for feedback should be easy and intuitive.
	4	Feedback should be unobtrusive to user and others.
	5	Feedback should be adjustable.

EMG = electromyographic.

Ces exigences sont réparties en trois sous-systèmes : les performances attendues sur l'analyse des signaux EMG, le contrôle de la prothèse et le retour d'information de celle-ci. On peut voir que des attentes très précises peuvent être dégagées.

Ainsi, l'augmentation de la qualité des protocoles expérimentaux serait un atout majeur pour les recherches futures qui pourraient alors recentrer leurs objectifs sur les attentes des utilisateurs.

2.6.2 Commande vocale

Un des domaines porteurs pour les prothèses myoélectriques serait le développement de la reconnaissance vocale pour le contrôle des tâches [50], [14]. La commande vocale pourrait, par exemple, permettre de réaliser des actions à la demande comme boire de l'eau ou couper sa viande. Il faudrait alors créer des commandes vocales pour activer des mouvements préenregistrés. Le principe est similaire aux déclencheurs par contractions des prothèses commerciales, mais demande bien moins de concentration et de patience.

2.6.3 Contrôle par le mouvement des yeux

Le mouvement des yeux peut être utilisé dans beaucoup de domaines et il a déjà démontré de très bonnes performances [51]. Dans l'industrie des jeux vidéo notamment, des recherches sont en cours de développement et leurs études seraient bénéfiques au développement des prothèses. Car, il y a trop peu de patients amputés pour réaliser de telles recherches. Mais la perceptive de pouvoir diriger sa prothèse avec les yeux en utilisant une paire de lunettes, par exemple, semble être une option très pratique [14].

2.6.4 Informations audio

Des commentaires audio pourraient être utilisés afin de fournir des informations concernant les mouvements de la prothèse [52]. Les oreilles ont un avantage particulier comme entrées : elles différencient déjà les signaux en provenance de la gauche ou la droite. Nous pourrions imaginer qu'un mouvement sur la gauche de la prothèse serait associé à une subtile fréquence de fond qui pourrait rendre compte de l'angle de flexion du coude tandis qu'une deuxième fréquence rapporterait des changements d'une autre variable.

2.6.5 Utilisation des nerfs pour le retour de sensation

Des travaux sont en cours pour fournir des informations directement aux nerfs spécifiques à la tâche [53] (TMR : Targeted Muscle Reinnervation). Cette technologie ne sera sûrement pas appliquée avant longtemps, car les patients ne souhaitent pas subir de chirurgies supplémentaires. La stimulation nerveuse directe semble pourtant avoir un fort potentiel. Cependant, le système nerveux est très sensible à l'adaptation et l'application de cette stimulation est encore à un stade expérimental.

Beaucoup d'autres recherches sont en cours et les avancées technologiques dans le domaine des prothèses myoélectriques sont nombreuses. En octobre 2016, par exemple, une main artificielle contrôlée par le cerveau a été testée [54]. Des électrodes ont été implantées directement sur le cortex cérébral et un algorithme a pour but d'interpréter les informations provenant du cerveau.

Cet état de l'art sur les prothèses myoélectriques nous montre l'étendue des possibilités d'améliorations, ainsi que les points importants à travailler pour le développement des fonctionnalités des prothèses. Une des difficultés actuelles, qui en ressort, est de pouvoir effectuer une classification des signaux EMG en temps réel précise et un bon traitement (plusieurs canaux, bonne amplification, diminution des bruits). C'est un des fondements qui permettrait à l'utilisateur de réaliser des mouvements plus intuitifs à l'aide d'une prothèse myoélectrique. C'est sur cette constatation que le projet présenté dans les parties suivantes s'est basé.

3 Vue globale technique du projet

Les objectifs techniques principaux du projet sont de réaliser une carte d'acquisition et de traitement efficace des signaux EMG, d'établir un système permettant de prédire des mouvements de la main de façon optimale, et de développer une prothèse mécatronique. Le but étant de réaliser un nouveau système de détection de mouvements complexes de la main, tels que la flexion indépendante de chaque doigt ou des saisies variées, pour contrôler une prothèse mécatronique.

Ce système de prédiction devra être rapide afin de permettre une certaine fluidité des mouvements. De plus, nous souhaitons faire un prototype embarqué peu coûteux pour l'utilisateur. L'aboutissement de ce projet serait d'avoir une prédiction de mouvement plus intuitive pour les usagers afin de faciliter le contrôle d'une prothèse myoélectrique.

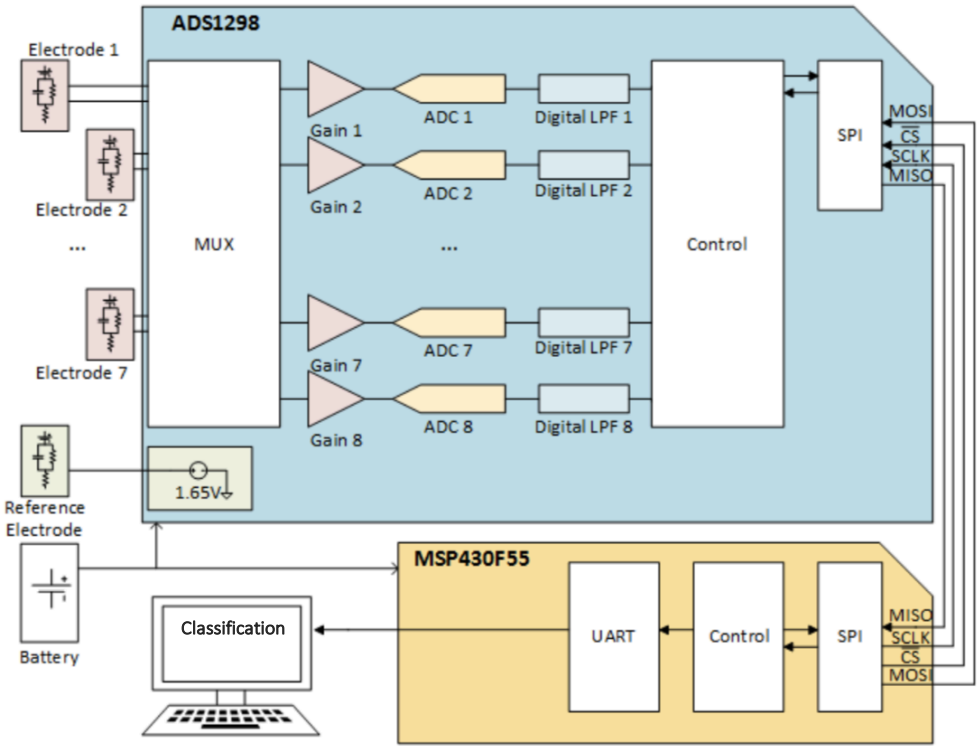

Figure 17 : Schéma-bloc du système de détection des mouvements de la main.

Dans la suite, nous aborderons le développement du système de détection des mouvements de la main. Le schéma-bloc de la Figure 17 présente le système dans son ensemble. Ce schéma-bloc a été construit au cours du projet afin de s'ajuster au mieux aux objectifs et à la problématique du projet : Comment développer un système de détection de mouvements complexes de la main, qui serait utilisé pour le contrôle d'une prothèse de main myoélectrique ? Les choix techniques des différents éléments de ce schéma ont été motivés par l'expertise de l'équipe du laboratoire de microsystèmes biomédicaux, ainsi que par la documentation et les moyens à notre disposition.

Les signaux EMG sont particulièrement bruités, d'amplitudes faibles et difficilement analysables à l'œil nu. D'où la nécessité de développer un système d'acquisition et de traitement efficace de ces signaux. Le premier objectif technique de ce projet a donc été de réaliser un tel système sur Printed Circuit Board (PCB) contenant un microcontrôleur MSP430F5529 et une puce spécialisée dans l'acquisition de signaux biologiques : l'ADS1298. Ce système permet la mesure et le traitement en temps réel de 7 canaux EMG provenant de 7 électrodes en entrées différentielles. Les électrodes ont également été conçues sur PCB et placées sur l'avant-bras de façon judicieuse. Le microcontrôleur assurera le contrôle des différents éléments de la carte d'acquisition, ainsi que la transmission des données à l'ordinateur de travail.

De plus, un algorithme de classification en temps réel de différents mouvements de la main (comme des flexions indépendantes des doigts ou des saisies variées) utilisant la méthode LDA a été testé sous Matlab. Des interfaces graphiques sous Matlab ont pu être développées afin de faciliter les interactions avec l'utilisateur. L'algorithme permet la détection de 13 mouvements différents avec une prédiction ajustée toutes les 192ms et une efficacité moyenne de 92,7%. Cet algorithme sera ensuite implémenté sur une Raspberry Pi Zero pour remplacer l'ordinateur de travail.

La prothèse est composée d'une main imprimée en 3D, de trois moteurs/réducteurs/encodeurs choisis judicieusement, de systèmes de transformation du mouvement par poulies et de trois contrôleurs.

La suite du mémoire détaillera ces différentes parties du projet ainsi que les choix techniques qui ont dû être faits.

4 Les électrodes et leurs positions

Dans cette partie, nous nous intéresserons aux muscles de l'avant-bras associés aux doigts de la main, afin de choisir un placement judicieux des électrodes de mesure. De plus, nous détaillerons les différents choix possibles de conception des électrodes ainsi que leur type et leur nombre. Ainsi, notre but sera de bien commencer ce projet en collectant des signaux EMG à des endroits stratégiques de l'avant-bras tout en ayant des électrodes efficaces pour la mesure EMG.

4.1 Les muscles de l'avant-bras

Nous allons, ici, nous pencher sur les différents muscles présents dans l'avant-bras et sur le placement des électrodes de mesure de notre système. C'est grâce à l'aide de Christian Brideau, du département de réadaptation de la faculté de médecine à l'Université Laval, que cette partie a pu être réalisée.

Il faut savoir qu'il existe une vingtaine de muscles dans l'avant-bras et il est donc nécessaire d'en prendre connaissance avant de placer les électrodes. Nous nous intéresserons plus particulièrement aux muscles fléchisseurs et extenseurs associés aux doigts de la main. Concernant les muscles extenseurs, nous pouvons trouver les muscles suivants :

- Extenseur commun des doigts
- Extenseur du petit doigt
- Long extenseur du pouce
- Court extenseur du pouce
- Extenseur de l'index

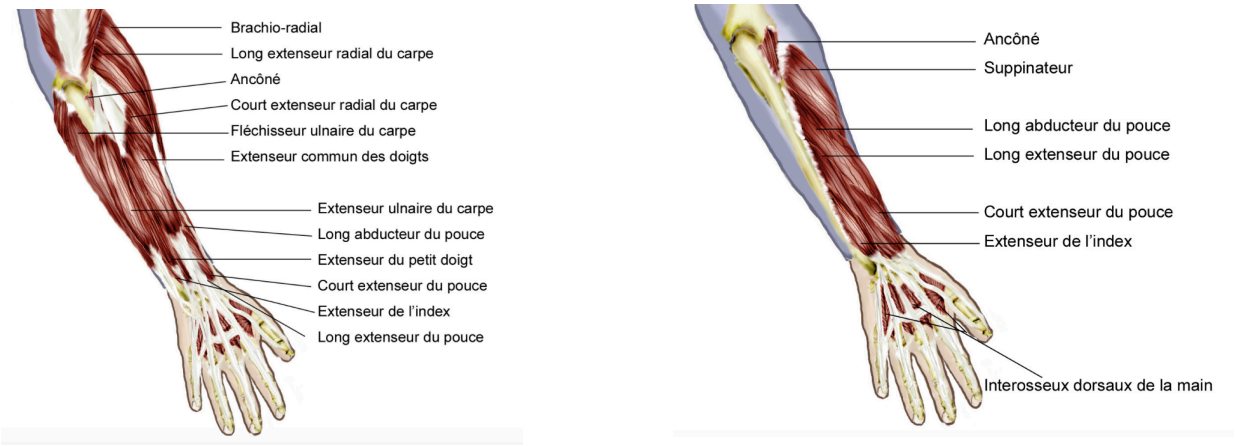

Figure 18 : Placement des muscles extenseurs dans l'avant-bras [Teitarc.com].

Pour les muscles fléchisseurs :

- Fléchisseur superficiel des doigts
- Fléchisseur profond des doigts

- Long fléchisseur du pouce

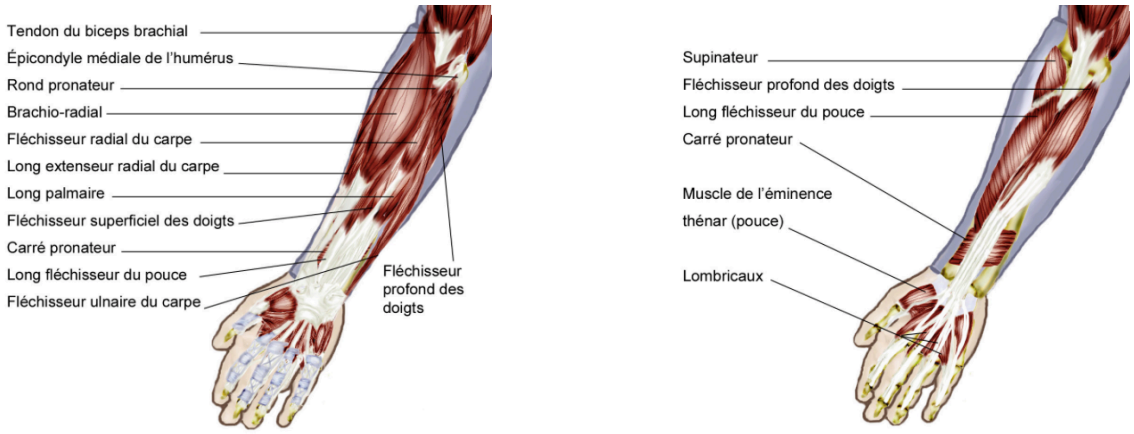

Figure 19 : Placement des muscles fléchisseurs dans l'avant-bras [nospot.org].

Ainsi, si nous plaçons une électrode sur la zone de l'avant-bras associée au muscle extenseur de l'index, nous pourrons mesurer une plus importante activité musculaire lors de l'extension de l'index ou de la contraction de ce muscle. La partie postérieure de l'avant-bras regroupe les muscles d'extensions (*extensors*) tandis que la partie antérieure les muscles de flexions (*flexors*).

Pour positionner correctement les électrodes sur les muscles de l'avant-bras, il faut d'abord se placer environ au 1/3 de l'avant-bras en partant du poignet, afin de s'éloigner des tendons présents au niveau du poignet. Les muscles fléchisseurs sont plutôt faciles d'accès, car les muscles fléchisseurs des doigts (*flexor digitorum sublimis*) et du pouce (*flexor pollicis longus*) sont présents en surface. Par contre, pour les muscles extenseurs, le muscle commun des doigts (*extensor digitorum commun*) est présent en surface, mais est très petit, les muscles associés à chaque doigt étant plus en profondeur.

Nous pouvons effectuer une flexion ou une extension des doigts afin d'identifier des zones de fortes intensités de contraction (forte activité musculaire) et placer les électrodes de façon judicieuse. La Figure 20 nous montre le placement choisi pour les électrodes.

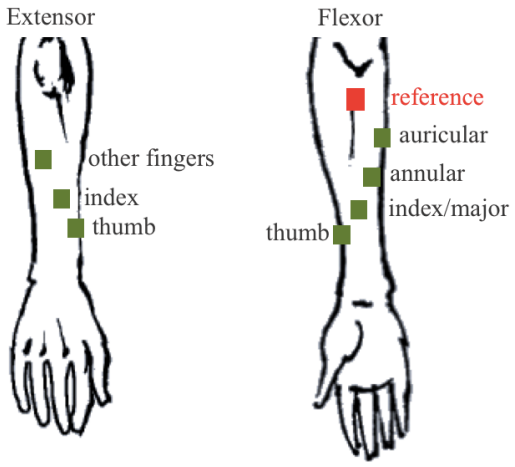

Figure 20 : Placement des 7 électrodes sur l'avant-bras, plus une de référence.

4.2 Choix du type d'électrode

Dans la littérature, différents types d'électrodes sont utilisés pour les prothèses myoélectriques. Certains chercheurs utilisent des électrodes de type humide comme les Ag/AgCl [34]. Ces électrodes ont la particularité, grâce à une solution Ag/AgCl en surface, de fournir des signaux de qualité. Malheureusement, ce type d'électrode ne peut être envisagé sur le long terme, car elles doivent continuellement être changées dû à l'oxydation rapide de la solution utilisée.

Des électrodes dites sèches peuvent également être utilisées. Un matériau conducteur est alors utilisé et permet d'enregistrer des signaux EMG. C'est ce type d'électrode qui est le plus souvent utilisé avec les prototypes ou même avec les prothèses commerciales. Nous n'avons pas choisi d'utiliser des électrodes commerciales, avec un circuit de traitement intégré à celle-ci, notamment en raison du prix des produits. Le but étant ici de développer un système peu coûteux pour l'utilisateur. De plus, nous voulions réaliser des électrodes personnalisées.

Des fibres conductrices très fines et souples sont développées par certains chercheurs pour enregistrer des signaux physiologiques [55]. Ces fibres peuvent, par exemple, être placées dans des vêtements (manches de t-shirt, de pull…) afin de faciliter leurs utilisations. L'avantage est la praticité d'utilisation, mais les résultats doivent encore être développés afin de montrer l'efficacité de ces fibres. Dans le même esprit, il existe des tissus conducteurs pouvant être utilisés pour enregistrer des signaux EMG [56].

On peut également utiliser des bracelets ou matrices d'électrodes, contenant un grand nombre d'électrodes, qui s'enroulent autour de l'avant-bras, par exemple le Myo Armband qu'on peut trouver dans le commerce. Ces bracelets fournissent beaucoup de signaux EMG à traiter et demandent un système possédant une grande puissance de calcul afin d'utiliser les données reçues. Nous avons préféré éviter ces matrices d'électrodes et privilégier un plus faible nombre d'électrodes placées judicieusement.

Nous avons choisi, pour notre étude, des électrodes sèches faites sur des PCB et testées afin de choisir les meilleures propriétés possible.

4.3 Conception des électrodes

Pour fabriquer des électrodes sèches, différents matériaux peuvent être utilisés comme du cuivre ou de l'or. Deux surfaces de contact du matériau conducteur doivent être espacées avec un matériau non conducteur afin d'enregistrer des signaux différentiels.

Plusieurs paramètres sont importants dans le design des électrodes :

- Le matériau utilisé pour les surfaces de contact conductrices (or, cuivre…etc.).
- La forme des surfaces de contact (rectangulaires, carrées, rondes…etc.).

- La superficie des surfaces de contact.
- L'espacement entre les surfaces de contact.

Ces paramètres vont influencer les signaux obtenus en sortie des électrodes, et notamment jouer un rôle important sur la qualité des signaux enregistrés. Nous avons alors décidé de concevoir plusieurs électrodes, en modifiant certains des paramètres, afin de déterminer la configuration la plus performante.

Les premières électrodes ont été développées avec des surfaces de contact en cuivre. L'inconvénient majeur du cuivre est son oxydation relativement rapide : au bout d'une journée, les électrodes placées sur la peau fournissent des signaux qui ne possèdent plus les mêmes propriétés qu'au matin. Ainsi, ce design a été abandonné (voir Figure 21).

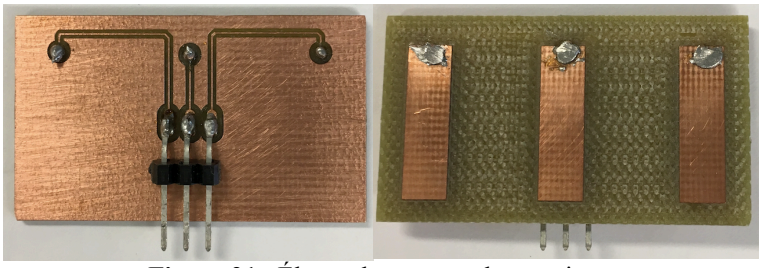

Figure 21 : Électrodes avec pads en cuivre.

De nouvelles électrodes ont été fabriquées (voir Figure 22) avec, cette fois, des surfaces de contact en or. Nous avons testé plusieurs formes de surfaces de contact (carrées, rectangulaires, rondes) et constaté que la forme ne jouait pas un rôle déterminant dans la qualité des signaux obtenus. Seule la superficie est réellement importante. Les électrodes avec des surfaces de contact rondes (Figure 22 (d) et (e)) ayant une superficie plus faible que l'électrode avec des surfaces de contact carrées (Figure 22 (f)), les signaux possèdent une amplitude de sortie plus faible. De même, les électrodes avec des surfaces de contact rectangulaires (Figure 22 (a) et (c)) enregistrent des signaux avec une amplitude plus forte que pour les carrées (Figure 22 (f)).

Les électrodes à surfaces de contact rectangulaires ont été choisies pour leurs bonnes amplitudes de sortie.

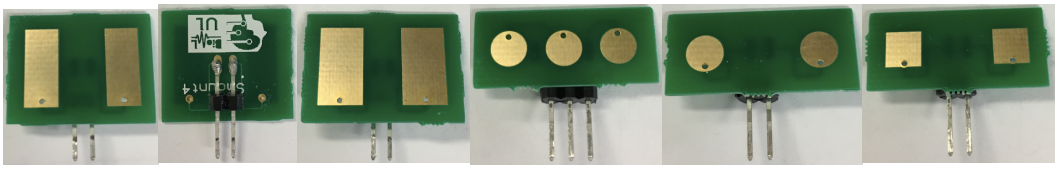

Figure 22 : Électrodes avec pads en or. (a) Électrode à pads rectangulaires, (b) Verso de l'électrode à pads rectangulaires, (c) Électrode à pads rectangulaires plus larges, (d) Électrode à 3 pads ronds (un en plus pour la référence), (e) Électrode à 2 pads ronds, (f) Électrode à pads carrés.

Nous avons également pu constater que l'espacement entre les surfaces de contact jouait sur le bruit des signaux de sortie. Plus l'espacement est faible, plus le bruit en sortie est grand. Il a donc fallu déterminer un espacement acceptable afin de minimiser le bruit tout en ayant des électrodes de faibles tailles. En effet, nous voudrions pouvoir placer un certain

nombre d'électrodes sur l'avant-bras. Pour cela, il ne faut pas que les électrodes soient trop encombrantes, ainsi une taille acceptable pour le contour du PCB (électrode) a été fixée. Les électrodes à surfaces de contact rectangulaires (a) et (c) ont été testées et comparées. L'électrode (a) a été choisie, car le bruit des signaux était moins important que pour la (c), malgré l'amplitude des signaux plus forte pour la (c). De plus, les électrodes sont volontairement plus longues que larges afin de pouvoir les placer dans la continuité du muscle (21 x 18 mm^2). Le design des électrodes a été fait sous Altium Designer, un logiciel spécialisé dans la conception des PCB.

4.4 Nombre d'électrodes utilisées et placement sur l'avant-bras

Après avoir choisi le design des électrodes (Figure 22 (a) et (b)), nous avons décidé du positionnement et du nombre d'électrodes à utiliser. La Figure 20 montre le placement des électrodes, choisi en fonction des muscles accessibles. Quatre électrodes sont consacrées aux muscles fléchisseurs et trois aux extenseurs. Le nombre d'électrodes a été déterminé selon la surface des muscles à couvrir : la surface des muscles fléchisseurs sur l'avant-bras est plus grande que les muscles extenseurs. Il aurait donc été inutile de mettre plus de trois électrodes pour les extenseurs dans la mesure où cela suffit pour couvrir l'ensemble de la surface utile des muscles. La Figure 23 présente le placement réel des électrodes sur l'avant-bras.

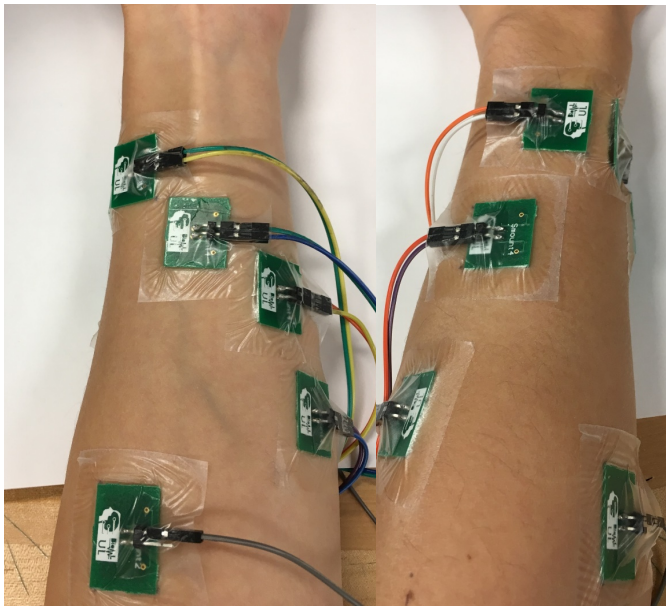

Figure 23 : Placement réel des électrodes sur l'avant-bras.

Cela résume les raisons d'un tel placement des électrodes sur l'avant-bras ainsi que le choix du type et du nombre d'électrodes utilisées.

5 Acquisition et traitement des signaux EMG

Dans cette partie, nous allons voir comment l'acquisition et le traitement des signaux EMG ont été réalisés, afin de pouvoir mettre en place, par la suite, un algorithme de détection des mouvements de la main en s'appuyant sur ces signaux traités.

5.1 Présentation générale des besoins

Ainsi, nous avons besoin de faire l'acquisition de 7 canaux EMG en temps réel (7 électrodes) afin de pouvoir avoir accès à chacun des muscles décrits dans la partie précédente. Les signaux EMG ont la particularité d'être très bruités et sujets à des perturbations extérieures. C'est donc pour cela que nous devons mettre en place un filtrage performant. La plage de fréquence utile des signaux se trouve entre 20 et 500Hz. De plus, ces signaux ont une amplitude relativement faible (entre -5mV et 5mV). C'est pourquoi une amplification sera nécessaire. Nous devrons réaliser un circuit permettant un filtrage entre 20 et 500Hz ainsi qu'une amplification. Pour terminer, nous aurons besoin d'un microcontrôleur pour gérer notre système et faire transiter les données de la partie d'acquisition à la réception sous Matlab. Les perturbations possibles du réseau électrique à 60Hz, seront filtrées par la suite sous Matlab.

Tout d'abord, nous présenterons le microcontrôleur choisi. Ensuite, nous aborderons les premiers tests réalisés sous Breadboard (platine d'expérimentation). Enfin, le design final du système d'acquisition et de traitement du projet sera analysé. Les données seront alors traitées par l'ordinateur de travail afin de réaliser un algorithme de classification des mouvements de la main.

5.2 Le microcontrôleur MSP430F5529

Pour ce projet, le microcontrôleur MSP430F5529 (Texas Instruments USA) a été choisi. Il possède un convertisseur analogique/numérique (ADC : Analog to Digital Converter), avec 12 bits de précision, pour convertir des données. De plus, il permet des communications UART (Universal Asynchronous Receiver-Transmitter) et SPI (Serial Peripheral Interface). Texas Instruments fournit notamment un kit d'utilisation très pratique que nous utiliserons au début du projet avec la Breadboard. Nous programmerons celui-ci à l'aide du logiciel Code Composer Studio 6.2.0 de Texas Instruments, dans un langage C.

Dans un premier temps, nous aurons besoin du microcontrôleur pour la conversion des données analogiques (provenant de la Breadboard) au format numérique en réalisant un échantillonnage à une fréquence de 5kHz. Cette fréquence a été choisie ici pour bien visualiser et analyser les signaux EMG, lors de cette phase de test sous Breadboard.

Dans un second temps, lorsque nous remplacerons la Breadboard par un système plus sophistiqué, ce microcontrôleur permettra la programmation des différents composants et l'échantillonnage sera fait à une fréquence de 1kHz, fréquence à la limite du théorème de

Shannon. La fréquence limite est choisie afin de limiter le nombre de données à traiter par la suite, car de nombreux calculs auront besoin d'être effectués sur les données dans un temps imparti. Enfin, nous aurons besoin d'établir une communication UART entre l'ordinateur et le microcontrôleur afin de transmettre les données à l'ordinateur. Cette communication pourra être remplacée plus tard par une communication sans fils Radio Fréquence (RF).

5.3 1ers traitements sur platine d'expérimentation

Dans un premier temps, la partie pour l'acquisition et le traitement analogique des données a été réalisée à l'aide d'une Breadboard. La Figure 24 présente le circuit utilisé pour un des canaux. Plusieurs exemplaires ont été mis en place afin de pouvoir mesurer en temps réel plusieurs canaux à la fois.

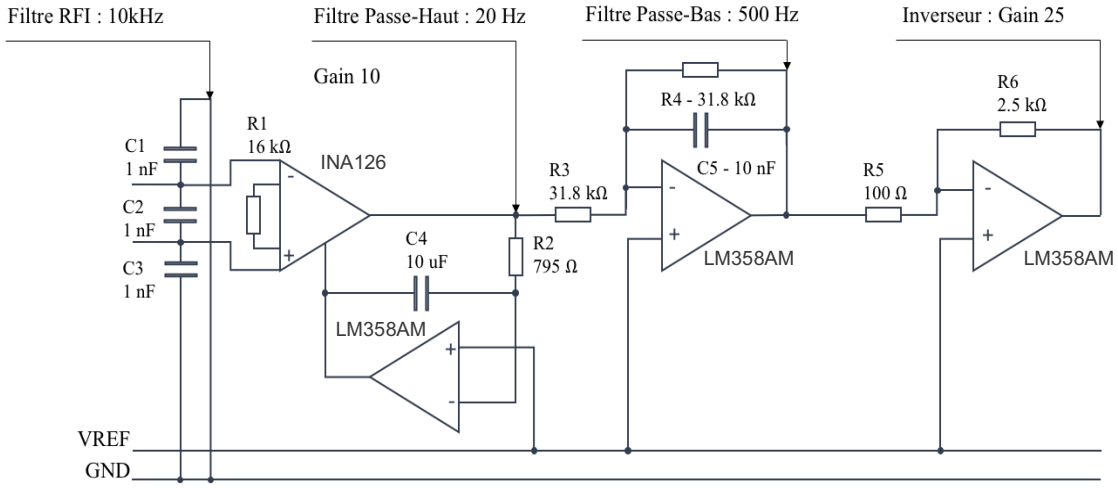

Figure 24 : Circuit réalisé pour un canal.

Tout d'abord, un filtre RFI permet de s'assurer que les hautes fréquences (supérieures à 10kHz) sont coupées. Le deuxième étage du circuit (composé d'un amplificateur d'instrumentation INA126, d'un amplificateur opérationnel LM358AM, de R1, C4, et R2) caractérise un filtre passe-haut avec comme fréquence de coupure 20Hz (les fréquences inférieures à 20Hz sont éliminées) et un gain de 10V/V. Le troisième étage (composé d'un LM358AM, de R3, R4 et C5) est un filtre passe-bas de fréquence de coupure 500Hz, donc toutes les fréquences supérieures à 500Hz sont coupées. Enfin le dernier étage du circuit (composé d'un LM358AM, de R5 et R6) est un inverseur qui permet de fournir un gain suffisant (Gain de 25V/V) pour analyser les signaux de façon correcte. De plus, un pont diviseur nous permet de créer une tension de référence à 1,25V afin de pouvoir ensuite travailler avec un microcontrôleur qui gère uniquement les tensions positives. Cette tension de référence est appliquée aux bornes positives des amplificateurs opérationnels (alimentés en 3,3V). Malgré l'ordre peu élevé des filtres utilisés, ce circuit est efficace pour obtenir des signaux facilement observables à l'œil nu. Il faudra par contre envisager des filtres numériques supplémentaires pour renforcer la qualité des signaux.

Ce circuit permet donc de conserver le signal utile de l'EMG : entre 20Hz et 500Hz. Il possède également un gain en tension de 250V/V afin d'amplifier suffisamment l'entrée pour la visualiser correctement.

Tableau 3 : Tableau des performances du circuit.

Paramètre	**Valeur**
EMG amplitude max (V)	0.005
EMG amplitude min (V)	0.0005
Gain en tension (V/V)	250
Bande passante (Hz)	20 – 500
Common Mode Rejection Ratio (CMRR en dB)	75

Nous avons testé ce circuit avec 2 canaux afin de voir son efficacité de filtrage. En sortie de notre circuit analogique sur Breadboard, le microcontrôleur MSP du kit convertit les données sous format numérique (ADC) à une fréquence d'échantillonnage de 5KHz puis les envoie par UART à l'ordinateur. Nous avons configuré l'envoi des données en une séquence de 32 bits contenant le canal 1 et le canal 2 en série. Voici le schéma bloc de l'acquisition par microprocesseur.

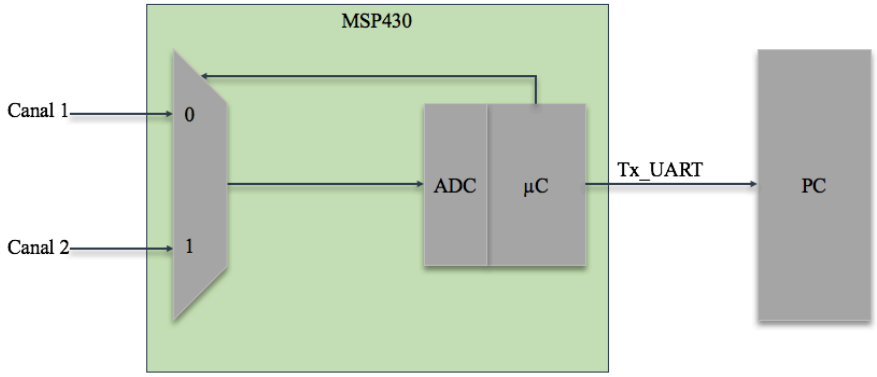

Figure 25 : Schéma bloc de l'acquisition par microprocesseur pour 2 canaux, lors de l'utilisation de la Breadboard.

La réception des données se fait sous Matlab. Dans un premier temps, le microcontrôleur envoyant les données en une séquence, nous devons trier les données reçues pour les placer dans deux variables distinctes représentant les deux canaux sous Matlab. Ces variables sont alors affichées en temps réel avec la fonction *stripchart* de Matlab. Des filtres supplémentaires ont été utilisés sous Matlab afin d'augmenter la qualité des signaux. Les avantages de ces filtres numériques, par rapport aux filtres analogiques, sont leurs facilités de conception, d'ajustement de leurs caractéristiques (ordres, fréquences de coupures…) et leurs stabilités dans le temps (pas d'usure de composants, etc.). Par contre, un des inconvénients serait qu'ils rajoutent une complexité de calcul. Les ordres des filtres ont été choisis de façon à optimiser le filtrage tout en ayant des temps de calcul acceptables pour ne pas ralentir l'exécution du programme. Voici ce qui a été implémenté :

- Filtre passe-bas de Butterworth d'ordre 10 avec une fréquence de coupure à 500Hz afin de couper les hautes fréquences non utiles.
- Filtre passe-haut de Butterworth d'ordre 8 de fréquence de coupure 20Hz pour couper la composante continue et les fréquences de 0 à 20Hz non voulues.
- Filtre coupe-bande de Butterworth d'ordre 4 avec une fréquence de coupure à 60Hz pour filtrer le bruit provenant de l'alimentation.
  La Figure 26 nous montre la qualité des signaux obtenus.

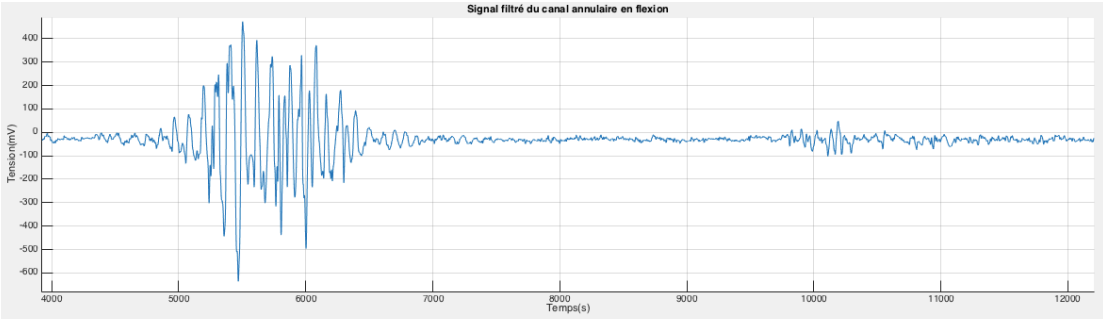

Figure 26 : Courbe du signal provenant du canal annulaire en flexion en sortie de la Breadboard avec l'acquisition sous Matlab.

Malgré la qualité acceptable de ces signaux, il a fallu se pencher sur la miniaturisation du système d'acquisition et de traitement. En effet, l'utilisation d'une Breadboard n'était pas envisageable pour la suite du projet. Il aurait notamment fallu reproduire les circuits de la Figure 24 en 7 exemplaires pour enregistrer les 7 canaux en temps réel. La Figure 27 nous donne un aperçu de la taille des circuits sous Breadboard. Nous avions besoin de réduire considérablement la taille du système. C'est pourquoi, dans la suite de ce projet, nous utiliserons le composant ADS1298 spécialisé dans l'acquisition des signaux biologiques.

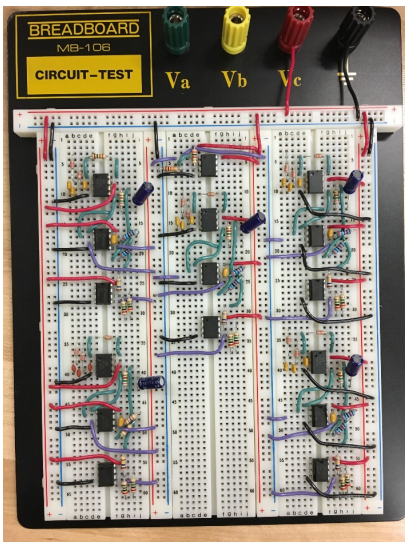

Figure 27 : Breadboard contenant 5 exemplaires du circuit de la Figure 24.

5.4 Remplacement de la platine d'expérimentation par l'ADS1298

Le composant ADS1298 est une puce électronique qui permet notamment l'acquisition de 8 canaux analogiques en temps réel. Il possède des convertisseurs analogiques/numériques et un filtrage performant. La Figure 28 nous montre l'ADS1298 (AFE : Analog Front-End) provenant de l'entreprise Texas Instruments USA.

Figure 28 : ADS1298 de Texas Instruments USA.

Le schéma-bloc de la Figure 17 présente le fonctionnement interne de l'ADS1298 pour 2 canaux d'entrées. Cette puce est composée de 8 entrées analogiques différentielles. Les signaux sont amplifiés par un gain programmable allant de 1 à 12 (nous choisirons 12 ici, car l'amplitude des signaux EMG est faible) et ensuite convertis par un convertisseur analogique/numérique Delta-Sigma sur 24 bits. Finalement, un filtre passe-bas numérique d'ordre 3 (sinus cardinal : sinc), filtre les signaux avec une fréquence de coupure programmable à -3dB. Celle-ci sera fixée à 524Hz, imposant une fréquence d'échantillonnage à 2kHz.

Les échantillons EMG sont codés sur 24 bits et ensuite réduits à 16 bits de précision pour diminuer la quantité de données à traiter. Dans la même optique de diminution du nombre d'échantillons, un échantillon sur deux est récupéré par le microcontrôleur MSP430F5529 à travers une communication SPI. Finalement, une communication UART a été implémentée, entre le microcontrôleur et l'ordinateur, permettant de récupérer, en temps réel, les échantillons associés aux 7 canaux d'entrées à travers Matlab. Une tension de référence, de 1,65V, est générée par l'ADS et imposée sur l'avant-bras de l'utilisateur à l'aide de l'électrode de référence. Les codes de programmation du microcontrôleur peuvent être retrouvés dans l'Annexe B.

Tableau 4 : Tableau récapitulatif des paramètres de l'ADS1298.

Paramètre	**Valeur**
Nombre d'entrées analogiques différentielles	8
Gain en tension programmable (V/V)	1 – 12 (12 sélectionné)
Précision du convertisseur Analogique/Numérique (ADC) Delta-Sigma (bits)	24
Fréquence de coupure (-3dB) du filtre passe-bas numérique sinc d'ordre 3 (Hz)	524
Fréquence d'échantillonnage (Hz)	2 000
Tension de référence (V)	1,65

5.5 Développement d'un PCB pour l'acquisition et le traitement des signaux EMG

Ainsi, connaissant les composants essentiels à l'acquisition et au traitement de nos 7 canaux EMG, un PCB sous Altium Designer a été développé. La Figure 29 montre ce PCB et les différentes parties de celui-ci. Cette partie s'est inspirée des articles [57], [58], [59], [60].

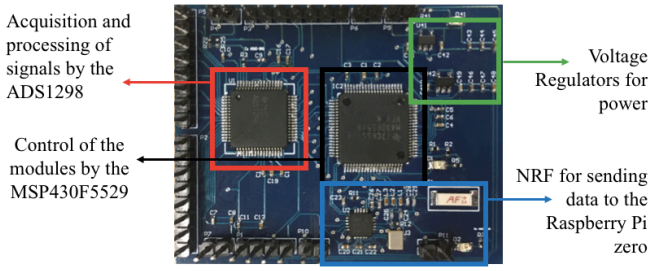

Figure 29 : Carte d'acquisition et de traitement des signaux EMG (56 x 45 mm^2).

L'acquisition des signaux analogiques et leurs traitements se font par l'ADS. La programmation de celui-ci passe par le MSP430F5529 en communication SPI. Le MSP récupère alors de l'ADS une donnée sur deux à traiter au format numérique sur 24 bits. Il les réduit à 16 bits avant de les envoyer par UART à l'ordinateur. De plus, deux régulateurs de tension permettent de créer une alimentation numérique et une analogique séparées pour l'ADS. Un module de transmission sans fils RF a été ajouté au PCB afin, au besoin, de communiquer à distance avec une Raspberry Pi. Celle-ci remplacera par la suite l'ordinateur afin de réaliser un système totalement embarqué. Toute la microsoudure de cette carte a été réalisée manuellement au laboratoire de microsystèmes biomédicaux de l'Université Laval.

Cette carte d'acquisition sera alimentée par une batterie LiPo (Lithium Polymère) (Figure 30) de capacité 650mAh à 3,7V et d'une puissance de 2,4W. Cette batterie permettra une très bonne autonomie du système d'acquisition (une centaine d'heures).

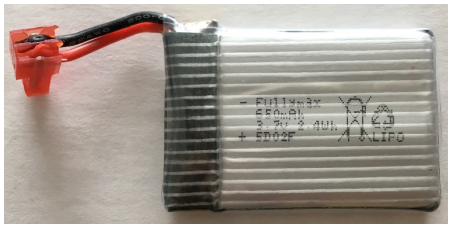

Figure 30 : Batterie LiPo.

5.6 Réception des données et traitement numérique des signaux EMG sous Matlab

Ainsi, le microcontrôleur va envoyer les données par UART à l'ordinateur. Matlab doit alors s'occuper de synchroniser la réception des données afin de pouvoir travailler en temps réel. Tout d'abord, il faut savoir que le microcontrôleur attend d'avoir des paquets de 450 données, avant d'envoyer le tout par UART. Dans ce paquet, nous avons 32 échantillons par

canal ainsi qu'un bit de début et de fin. Sauf qu'ici les échantillons sont codés sur 16 bits, or le buffer utilisé pour l'envoi des données par UART se trouve être sur 8 bits. Donc un échantillon sera décomposé en 2 paquets de 8 bits pour être envoyé en 2 fois. Ce qui, finalement, fait des paquets de : $(32\times2\times7)+2=450$ données (une donnée étant sur 8 bits). Le Baud Rate utilisé entre le microcontrôleur et l'ordinateur est de 921 600 bits/seconde. Nous l'avons choisi relativement grand afin de pouvoir envoyer toutes les informations voulues dans les temps.

Nous devons mettre en place une routine Matlab qui se fera de façon récurrente. Dans notre cas, étant donné l'algorithme que nous allons faire, il va falloir correctement gérer les temps de calcul. Récupérer les données par UART reste une opération assez longue. Il vaut mieux ainsi, récupérer en une seule fois un paquet de données important plutôt que de récupérer en plusieurs fois des petits paquets. Nous allons donc attendre d'avoir 6 paquets de 450 données disponibles dans le buffer interne de l'ordinateur, avant d'effectuer notre routine Matlab. Ce qui nous donnera en tout, des fenêtres de travail de 192 échantillons/canal disponibles. L'algorithme sera codé de telle sorte qu'à chaque 5ms, nous regarderons dans le buffer interne de l'ordinateur si 6 paquets de 450 données sont disponibles. Si c'est le cas, la routine peut commencer à effectuer ses opérations. Les 6 paquets seront, en fait, disponibles toutes les 192ms, car un échantillon/canal est récupéré à une fréquence de 1kHz. Ainsi, nous traiterons des fenêtres de 192ms sous Matlab et pourrons implémenter un algorithme de classification qui nous donnera une prédiction toutes les 192ms (pour respecter le critère de temps réel en robotique, la latence doit être inférieure à 300ms [27], afin d'avoir un mouvement fluide).

L'algorithme Matlab récupère alors les échantillons en vérifiant les bits de débuts et de fins, et les trie en 7 variables séparées correspondantes aux données de chacun des canaux. Chaque échantillon est alors reformé en utilisant le complément à 2 afin de recomposer les 2 moitiés d'échantillon en un. On retrouve ensuite l'amplitude d'origine (en gardant le gain de 12) des échantillons en utilisant la plage de fonctionnement des ADC dans l'ADS.

Pour terminer, nous avons également décidé d'établir des filtres numériques en temps réel sous Matlab. Des ordres peu élevés ont suffi ici, étant donné le filtrage préalable, très performant, effectué par l'ADS. Ainsi, le filtrage a pu être optimisé tout en ayant des temps de calcul acceptables pour ne pas ralentir l'exécution du programme, qui doit se faire en moins de 192ms.

- Un filtre passe-bande Infinite Impulse Response (IIR) de Butterworth du second ordre afin d'avoir une bande-passante entre 20 et 500Hz.
- Un filtre coupe-bande IIR du second ordre pour filtrer le 60Hz pouvant provenir de l'alimentation.

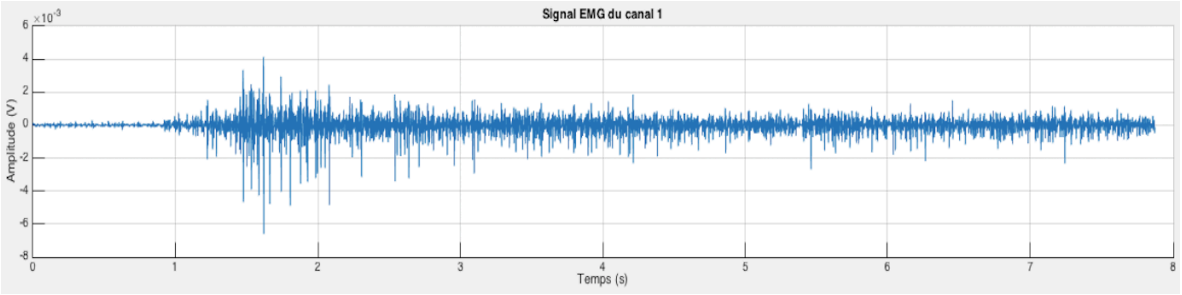

Figure 31 : Courbe du signal provenant du canal du pouce en flexion en sortie de la carte d'acquisition et après filtrage sous Matlab.

L'opérateur TKE [35] des signaux EMG sera également calculé afin de mettre en place un « onset detection », et ainsi détecter la présence d'un mouvement ou non. En effet, le TKE permettra de déterminer le début du mouvement. Nous verrons plus tard que nous aurons besoin d'enregistrer des mouvements sur une certaine durée. Pour cela, la valeur moyenne et l'écart-type du TKE sont calculés. Le seuil de mouvement est alors pris comme étant la moyenne du TKE ajoutée à 5 fois l'écart-type du TKE. Le TKE permet notamment d'amplifier la partie utile du signal tout en minimisant le bruit [61]. On peut le calculer avec la formule (1).

Cela conclut la partie sur l'acquisition et le traitement des signaux EMG. La carte électronique, réalisée sur un PCB, nous permet une acquisition propre, ainsi qu'une communication avec l'ordinateur de travail sous lequel l'algorithme Matlab de détection de mouvements a été développé.

6 Algorithme de classification des signaux EMG sous Matlab

6.1 Vue globale et objectifs

Dans cette partie, l'algorithme de classification des signaux EMG va être détaillé. Notre objectif est de classifier en temps réel 13 mouvements complexes de la main. La Figure 32 présente les 13 mouvements en question.

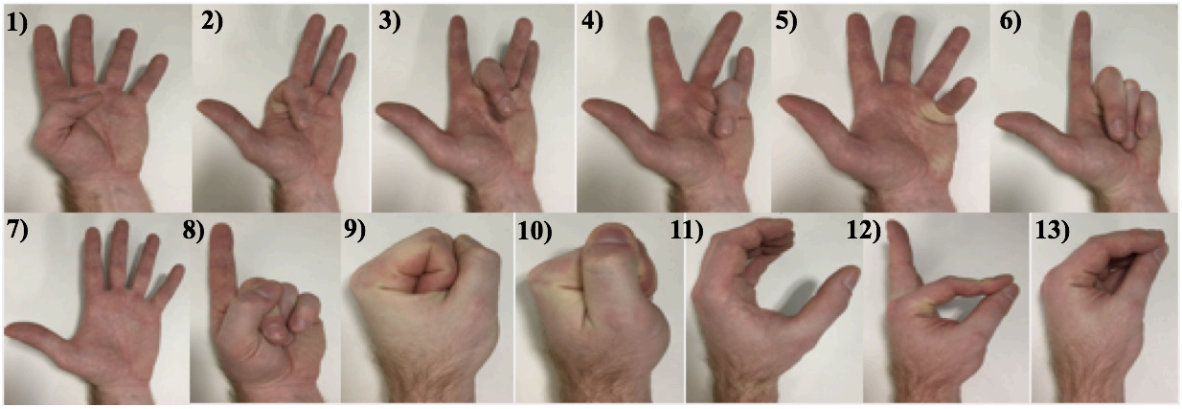

Figure 32 : Les différents mouvements (classes) à reconnaître. 1) Flexion du pouce, 2) de l'index, 3) du majeur, 4) de l'annulaire, 5) de l'auriculaire, 6) des 3 doigts, 7) main ouverte, 8) pointer l'index, 9) main fermée, 10) prise latérale, 11) prise cylindrique, 12) pince simple et 13) pince complexe.

Pour assurer une certaine fluidité de mouvement et respecter le critère de temps réel, il faut s'assurer qu'un nouvel ordre soit envoyé dans un temps inférieur à 300ms [27]. Ainsi, l'algorithme doit être rapide d'exécution, car on attend une prédiction toutes les 192ms. Toutes les tâches doivent donc être exécutées en moins de 192ms afin d'éviter d'avoir de la latence.

Pour la classification, la méthode LDA sera utilisée et ses paramètres discriminants (*features*) seront choisis judicieusement. Pour chaque sujet utilisant la prothèse, une base de données (*database*) sera créée. Cette *database* contiendra des observations des *features* pour chacun des mouvements à classifier. C'est grâce à cette base de données que le système pourra s'entraîner et par la suite prédire le mouvement en temps réel. Il est important que la *database* soit refaite pour chaque nouvel utilisateur, car les signaux EMG ont des caractéristiques propres à chacun.

Afin de créer facilement une nouvelle base de données, une interface graphique sera développée sous Matlab. Dans un second temps, une deuxième interface sous Matlab servira pour la prédiction en temps réel.

6.2 La méthode *Linear Discriminant Analysis*

Nous avons choisi de nous pencher sur la méthode LDA [36] pour la classification des mouvements. En effet, cette méthode à la particularité d'être très rapide d'exécution (de l'ordre de 10ms sous Matlab avec un processeur Intel Core i5 cadencé à 1,3GHz), la vitesse

étant un critère essentiel pour du temps réel. De plus, la méthode ne demande pas beaucoup de puissance de calcul et nous permettra de l'utiliser sur un système embarqué par la suite (faible coût en énergie). C'est une méthode précise et connue pour sa robustesse au cours du temps. L'article [62] présente une autre méthode de classification intéressante.

Cette méthode se base sur un ensemble de paramètres discriminants choisis judicieusement afin de caractériser le signal EMG. Pour chaque mouvement à conserver dans la base de données, les *features* du mouvement sont calculés et enregistrés. On recherche par la suite, dans la base de données, les paramètres discriminants les plus proches de ceux du mouvement à classifier.

De manière plus technique, la méthode LDA se déroule en *n* dimensions (*n* étant le nombre de *features* choisi). Elle crée des axes qui maximisent la séparation des classes (mouvements) entre elles. Ces axes dépendent de toutes les dimensions et donc des *features*. C'est ainsi que l'on parvient à classifier les différents mouvements. La Figure 33 est un exemple de séparation de classes pour deux paramètres discriminants : l'amplitude maximum du canal 2 et celle du canal 3. Les données de chaque classe sont représentées par les marquages ponctuels de couleurs et les axes correspondent à la séparation entre les différentes classes. On peut constater qu'une bonne séparation est faite entre les classes.

La distance de séparation $(m_i - m)$ entre les différentes classes peut se calculer selon cette formule :

$$(m_i - m)^2 = W^T(\mu_i - \mu)(\mu_i - \mu)^T W \quad (2)$$

Où $m_i = W^T\mu_i$, représente la projection (sur l'espace dimensionnel du LDA) de la moyenne des échantillons EMG de la classe i et $m = W^T\mu$, la projection de la moyenne totale sur toutes les classes. W correspond à la matrice de transformation de la méthode LDA. μ_i est la moyenne des échantillons EMG de la classe i et μ la moyenne totale des échantillons EMG sur l'ensemble des classes. Le développement mathématique de la méthode LDA est présenté plus en détail dans [36].

Deux fonctions sous Matlab sont utilisées pour réaliser cette méthode. La fonction *fitcdiscr* qui utilise la *database* pour effectuer la discrimination linéaire entre les classes. Et la fonction *predict* qui prend le résultat de la discrimination et envoie un mouvement (vecteur de *features*) à classifier.

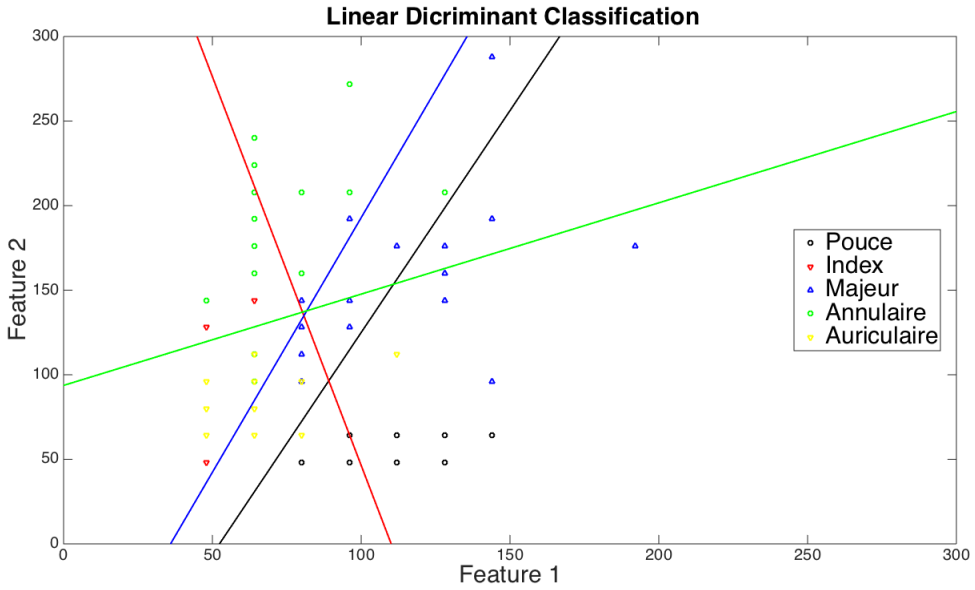

Figure 33 : Exemple de séparation de classes par la méthode LDA. Deux *features* (correspondant aux deux axes) sont utilisés ici : l'amplitude maximum du canal 2 en abscisse et celle du canal 3 en ordonnée. Les différentes classes ici sont : la flexion du pouce en noir, la flexion de l'index en rouge, la flexion du majeur en bleu, la flexion de l'annulaire en verte et celle de l'auriculaire en jaune. L'axe noir correspond à la séparation entre les classes du pouce et de l'index, l'axe rouge à la séparation de l'index et du majeur, en bleu du majeur et de l'annulaire, et en vert de l'annulaire et de l'auriculaire.

6.3 Le choix des paramètres discriminants utilisés

Concernant le choix des *features*, utilisés avec la méthode LDA, deux types peuvent être mis en place : des *features* temporels ou fréquentiels. Les derniers demandent une plus grande complexité de calcul et ne sont pas plus discriminants que les *features* temporels. Leur seul avantage est d'être moins perturbés par le bruit et la qualité des signaux enregistrés. C'est pourquoi nous choisirons essentiellement des paramètres discriminants temporels hormis un fréquentiel. Il est très important de comprendre l'importance du choix des *features* : c'est eux qui vont caractériser les signaux EMG provenant des 7 canaux d'électrodes. C'est donc grâce à un choix avisé de ceux-ci que nous pourrons augmenter le pouvoir discriminant de l'algorithme de classification. Il faut, de plus, prendre en compte que certains *features* sont plus adaptés à une méthode de classification plutôt qu'à une autre. Les articles [32], [33] et [34] nous ont permis de faire ce choix des paramètres discriminants. Ils présentent les formules mathématiques et l'efficacité de plusieurs *features* avec différents classifieurs. Les signaux EMG variant d'un utilisateur à l'autre, certains *features* pourraient voir leurs pouvoirs discriminants varier d'un usager à l'autre, c'est aussi pourquoi plusieurs paramètres, avec différentes caractéristiques, ont été retenus.

Voici la liste des paramètres discriminants utilisés, tous seront calculés sur une certaine plage des signaux EMG, dont nous parlerons dans la partie suivante :

- Mean Absolute Value (MAV) (paramètre temporel) [63] : ce *feature*, très populaire dans la littérature, correspond à la moyenne des valeurs absolues de l'amplitude.

- Zero Crossing (paramètre temporel) [63] et [64] : il mesure une certaine information fréquentielle d'un signal EMG dans le domaine temporel. C'est en fait le taux de changement de signe d'un signal en fonction d'un seuil que l'on choisit. Ce seuil permet notamment d'éviter de prendre le bruit en considération. On le choisira à 0,5mV, car nous utilisons un gain de seulement 12 avec notre système, ce qui donne une plage des signaux EMG entre -60mV et 60mV environ. Le même seuil sera également utilisé dans plusieurs autres *features*.

- Slope Sign Change (paramètre temporel) [63] : ce paramètre mesure également une information fréquentielle dans le domaine temporel. Il calcule le nombre de fois où la pente du signal change de signe en fonction du seuil.

- Waveform Length (WL) (paramètre temporel) [63] : comme son nom l'indique, il correspond à la longueur de la forme d'onde EMG et permet une mesure de la complexité du signal EMG. Voici sa formule :

$$WL = \sum_{i=1}^{N-1} |x_{i+1} - x_i| \quad (3)$$

Où x_i représente l'échantillon i du signal EMG et N correspond à la longueur du signal EMG.

- Wilson Amplitude (WAMP) (paramètre temporel) [63], [64] et [65] : ce paramètre permet de mesurer une information fréquentielle. Il est en lien direct avec les actions de potentiels et la force de contraction d'un muscle. Voici sa formule :

$$WAMP = \sum_{i=1}^{N-1} f(|x_i - x_{i+1}|) \quad (4)$$

Avec f(x) = 1 si x $\geq$ seuil et 0 sinon. Le seuil est choisi de façon à rendre indépendant le paramètre discriminant au bruit et à la fluctuation des faibles tensions. On le fixera a 0,5mV.

- Coefficients Autorégressifs d'ordre 4 (paramètres temporels) [63] et [64] : ces coefficients caractérisent un modèle de prédiction qui décrit chaque échantillon EMG comme une combinaison linéaire des échantillons précédents plus un bruit blanc. Voici la formule utilisée, les coefficients a_p seront ceux utilisés comme paramètres :

$$x_i = \sum_{p=1}^{o} (a_p . x_{i-p} + \omega_i) \quad (5)$$

Avec o=l'ordre, suggéré à 4 le plus souvent, xi l'échantillon i du signal EMG, ω_i un bruit blanc ajouté et a_p les coefficients autorégressifs. La méthode d'estimation de Yule-Walker sera utilisée pour le calcul de ces coefficients [66].

- Skewness (SK) (paramètre temporel) [67] : il permet de mesurer l'asymétrie de la distribution des probabilités du signal EMG réel.

$$SK = \mathbb{E}\left[\left(\frac{X-\mu}{\sigma}\right)^3\right] \quad (6)$$

Avec $\mathbb{E}$ l'espérance, X le signal EMG, μ la moyenne du signal EMG et σ son écart type.

- Integrated EMG (IEMG) (paramètre temporel) [33] et [67] : ce *feature* est la somme des valeurs absolues de l'amplitude.

- Hjorth Activity (HA) (paramètre temporel) [68] : il permet la mesure de la variance du signal. Il indique notamment la surface de la puissance spectrale dans le domaine fréquentiel.

$$\mathrm{HA} = \mathrm{var}(\mathrm{X}) \tag{7}$$

Avec $\mathrm{var}(\mathrm{X})$ la variance du signal EMG.

- Hjorth Mobility (HM) (paramètre temporel) [68] : ce paramètre correspond à la moyenne fréquentielle par un calcul temporel. Son calcul se base sur le paramètre précédent. Voici sa formule :

$$\mathrm{HM} = \sqrt{\frac{\mathrm{var}(\mathrm{d}(\mathrm{X}))}{\mathrm{var}(\mathrm{X})}} \tag{8}$$

Avec $\mathrm{d}(\mathrm{X})$ la dérivée première du signal EMG.

- Hjorth Complexity (HC) (paramètre temporel) [68] : il représente le changement de fréquence. Son calcul se base sur HM. C'est le ratio de la mobilité de la dérivée du signal sur la mobilité du signal.

$$\mathrm{HC} = \frac{\mathrm{HM}(\mathrm{d}(\mathrm{X}))}{\mathrm{HM}(\mathrm{X})} \tag{9}$$

- Mean Frequency (MNF) (paramètre fréquentiel) [33] : ce paramètre correspond à la fréquence centrale d'un signal. C'est en fait la somme des produits de la puissance spectrale et de la fréquence, le tout divisé par le total de la somme de la puissance spectrale. Voici sa formule :

$$\mathrm{MNF} = \frac{\sum_{\mathrm{j}=1}^{\mathrm{M}} \mathrm{f_j}.\mathrm{P_j}}{\sum_{\mathrm{j}=1}^{\mathrm{M}} \mathrm{P_j}} \tag{10}$$

Avec f la fréquence du signal EMG et P sa puissance spectrale.

Les paramètres discriminants ont été choisis en fonction de leur efficacité de discrimination des mouvements lorsqu'on utilise la méthode LDA. De plus, la plupart demandent très peu de temps de calcul ce qui est un critère essentiel ici. Le but est d'optimiser le pouvoir discriminant de l'algorithme de classification tout en ayant des temps de calcul assez faibles pour travailler en temps réel. Un compromis devra toujours être fait entre efficacité de classification et temps d'exécution.

6.4 Le découpage des fenêtres et la classification temps réel

Un algorithme de classification a donc été mis en place en utilisant la méthode LDA. Celui-ci peut être décomposé en deux points importants : la construction de la base de données et la prédiction en temps réel. La Figure 34 expose l'algorithme de classification.

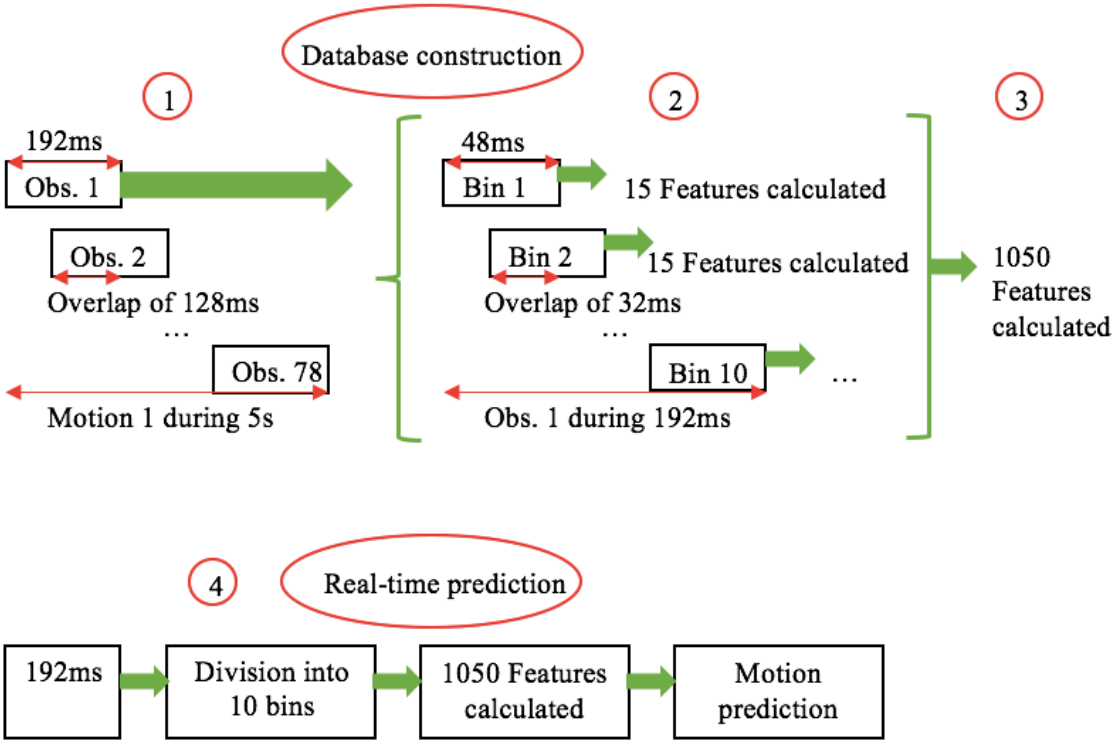

Figure 34 : Algorithme de classification.

Commençons par la partie sur la construction de la *database*. Cette base de données sera construite à partir des données EMG traitées en temps réel, mais le calcul des *features* ne se fera pas en temps réel. L'ensemble des *features* vu dans la partie précédente seront sélectionnés. Pour avoir une bonne efficacité de classification des mouvements, la *database* doit être suffisamment complète pour permettre la reconnaissance des mouvements. Il faudra donc enregistrer les 13 mouvements à classifier un certain nombre de fois. Nous avons choisi, ici, de faire 10 répétitions de chaque mouvement, maintenu sur une durée de 5s. Cela semble être un bon compromis pour avoir une bonne efficacité sans consacrer trop de temps à la construction de la base de données.

Notre but étant d'avoir une nouvelle prédiction sur le mouvement toutes les 192ms (pour le contrôle d'une prothèse en temps réel), il nous faut construire une base de données de sorte que chaque observation ait une taille de 192ms. Ainsi, les 5s de mouvement seront découpées en 78 fenêtres de 192ms qui se chevauchent (chevauchement de 128ms = 2/3 de 192ms).

Sur chacune de ces fenêtres, nous allons calculer un certain nombre de paramètres discriminants. Pour augmenter le nombre de *features* utilisés, une découpe de la fenêtre de 192ms sera faite pour produire 10 segments qui se chevauchent de 32ms. Sur chacun de ces

segments, nous calculerons les 15 *features* décrits plus haut, et ce, pour les 7 canaux EMG. En résumé pour la construction de la base de données, sur une observation de 192ms, nous calculerons au total 1050 *features* (10×15×7). Le Tableau 5 nous donne un aperçu de l'allure de la base de données. C'est cette matrice, ainsi que le vecteur contenant les classes associées à chaque observation, qui seront utilisés pour la création du modèle linéaire avec *fitcdiscr*.

Tableau 5 : Forme de la matrice de la base de données de taille 10140x1050. *Feature* 1 = MAV, *feature* 2 = Zero Crossing, *feature* 3 = Slope Sign Change, *feature* 4 = WL, *feature* 5 = WAMP, *feature* 6-9 = Coeff AR, *feature* 10 = MNF, *feature* 11 = Skewness, *feature* 12 = IEMG, *feature* 13 = Hjorth Activity, *feature* 14 = HM, *feature* 15 = Hjorth Complexity.

Features/ Observations	*Feature* 1 Canal 1 Segment 1	…	*Feat*. 1 Canal 1 Seg. 10	*Feat*. 2 Canal 1 Seg. 1	…	*Feat*. 15 Canal 1 Seg. 10	*Feat*. 1 Canal 2 Seg. 1	…	*Feat*. 15 Canal 2 Seg. 10	…	*Feat*. 15 Canal 7 Seg. 10
Obs. 1 pouce	…										
…											
Obs. 78 pouce											
…											
Obs. 780 pouce											
Obs. 1 index											
…											
Obs. 1 majeur											
…											
Obs. 780 prise latérale											

La deuxième partie importante de l'algorithme est la prédiction en temps réel qui doit être faite : toutes les fenêtres de 192ms seront analysées tour à tour. Cette fois, tout sera fait en temps réel : l'acquisition et le traitement des données EMG, et le calcul des paramètres discriminants. Pour avoir un vecteur d'observation comparable à la base de données, chaque fenêtre de 192ms sera découpée en 10 segments et nous calculerons les paramètres discriminants sur chacun de ces segments. Une fois ces calculs effectués, la fonction *predict* de Matlab déterminera le mouvement associé à cette fenêtre. Cette fonction *predict* se base sur le modèle créé avec *fitcdiscr* pour pouvoir donner un résultat. Elle calcule la réponse prédite par le modèle pour le vecteur de *features* donné en entrée. L'algorithme de classification en temps réel devra, s'exécuter en moins de 192ms pour éviter tout décalage. Son temps d'exécution, ici, est en moyenne de 145ms.

6.5 Développement d'une interface graphique Matlab pour la construction de la base de données

Afin de permettre aux utilisateurs de construire plus facilement une base de données, une interface graphique a été développée (Figure 35). Elle permet de lancer l'acquisition pendant le temps de notre choix, de visualiser les 7 canaux EMG, d'arrêter l'acquisition ou de la réinitialiser. Les codes peuvent être retrouvés dans l'Annexe A.

En général, une acquisition de 8s est choisie afin de laisser 3s à l'utilisateur pour initier le mouvement et le maintenir pendant 5s. On peut alors observer les signaux enregistrés, leurs transformées de Fourier rapides (FFT : Fast Fourier Transform), et leurs formes TKE. Il faut ensuite choisir le mouvement qui a été fait, le lieu de sauvegarde voulu et ce qu'on souhaite enregistrer : *features*, données filtrées, TKE ou toutes les données.

Afin de déterminer le début du mouvement effectué par l'utilisateur, et ainsi enregistrer 5s de mouvement, un seuil calculé à partir de la forme TKE est utilisé [69]. Pour cela, la valeur moyenne et l'écart type du TKE sont calculés à partir des 8s de données. Voici la formule des seuils établis pour chaque canal indépendamment :

$$\text{seuil\_canal\_1} = \text{moyenne\_TKE\_canal\_1} + (5 \times \text{ecart\_type\_TKE\_canal\_1}) \qquad (11)$$

Si l'un des canaux passe au-dessus de son seuil, alors de nouvelles variables sont créent contenant l'enregistrement des 5s de mouvement pour chaque canal. Ajouter l'écart type permet d'établir un seuil au-dessus des bruits potentiels.

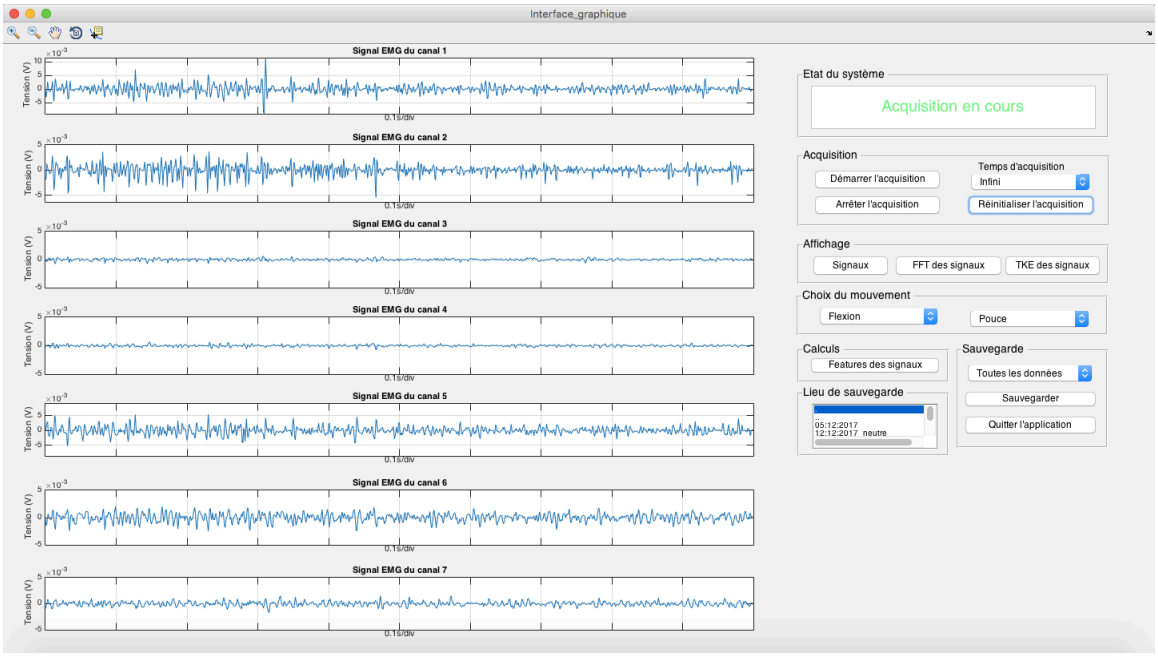

Figure 35 : Interface graphique pour la construction de la *database* lors d'une flexion du pouce.

En résumé, cette interface permet donc de choisir le mouvement à effectuer, puis d'enregistrer les paramètres discriminants correspondants au mouvement en question tout en ayant un retour visuel des signaux EMG en temps réel. Cette interface est facile d'utilisation et intuitive pour l'utilisateur. De plus, nous pourrons continuer à l'utiliser pour créer la base de données même après que le code en temps réel soit implanté sur Raspberry Pi. En effet, l'utilisateur pourra enregistrer ses mouvements à l'aide de son ordinateur puis la base de données sera directement récupérée par la Raspberry Pi qui construira le modèle linéaire pour la prédiction.

6.6 Développement d'une interface graphique pour la prédiction en temps réel

Une seconde interface graphique a été développée (Figure 36) pour avoir un retour visuel sur le mouvement prédit en temps réel. Nous allons également ajouter une calibration du mouvement neutre, main ouverte. Les codes peuvent être retrouvés dans l'Annexe A.

En effet, bien que le mouvement neutre existe dans notre classification, les perturbations extérieures influencent les signaux EMG et plus particulièrement le neutre. Ce qui implique que celui-ci est moins bien reconnu que les autres mouvements si les conditions extérieures évoluent au fil du temps. Or, nous voulons permettre l'utilisation de la même base de données le plus longtemps possible au cours du temps. Pour cela, une calibration supplémentaire très rapide permettra à l'utilisateur d'ajuster la reconnaissance du mouvement main ouverte.

On effectue donc la calibration sur 5s, afin d'établir des seuils et déterminer la présence d'un mouvement ou non en temps réel. Le maximum du TKE pour chaque canal est calculé sur la plage des 5s. Ce sont ces maximums qui nous serviront de seuils en temps réel. Lorsque les 192 valeurs de chaque canal sont inférieures aux seuils, nous sommes dans le cas du mouvement neutre et nous n'effectuons pas de classification. Si par contre, une seule des 192 valeurs d'un seul canal dépasse le seuil, alors la classification est enclenchée. Cela nous permet de gagner en robustesse de prédiction.

Avec cette interface, une base de données particulière peut être chargée sous Matlab et le modèle LDA, utilisé pour la prédiction, est calculé. L'interface graphique permet également à l'utilisateur de voir le résultat de la classification, ainsi que l'image du mouvement prédit. Une fluidité dans la classification a pu être constatée, car une prédiction a lieu toutes les 192ms. Une vidéo montrant le système de détection des mouvements de la main peut être trouvée en suivant ce lien : *https://youtu.be/BgPJ_FCgmrk*.

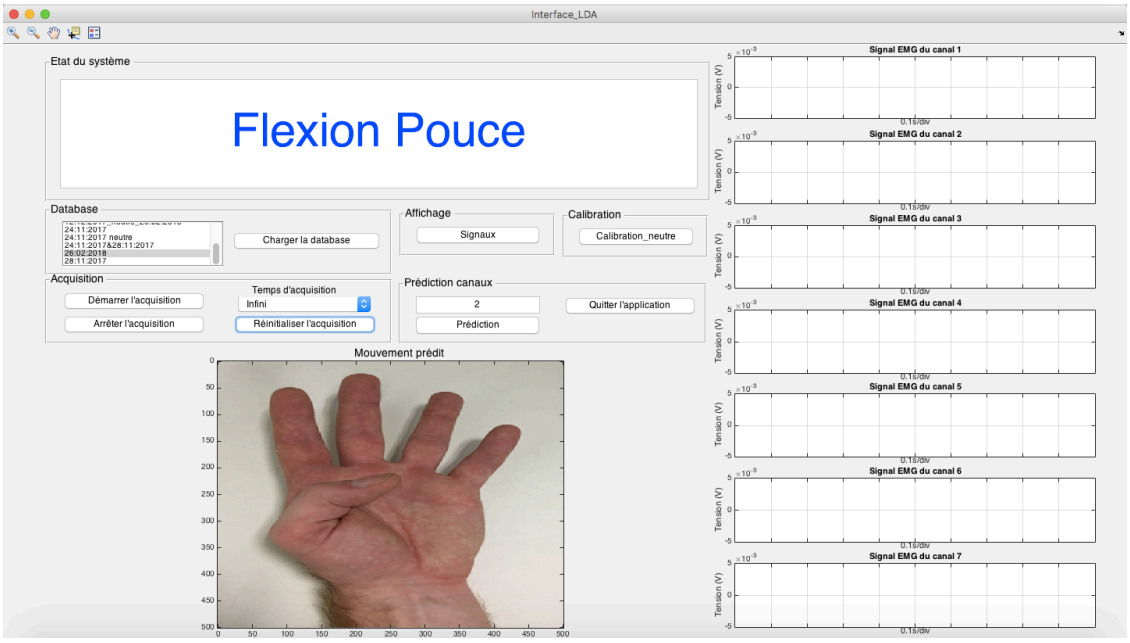

Figure 36 : Interface graphique de prédiction lors d'une flexion du pouce.

7 Développement d'un système embarqué sur Raspberry Pi Zero

Dans cette partie, le portage du code Matlab sur Raspberry Pi Zero va être abordé. En effet, jusque-là, un ordinateur de travail était nécessaire pour avoir accès à la prédiction en temps réel. Désormais, nous pourrons utiliser une Raspberry Pi, qui remplacera l'ordinateur de travail, et ainsi rendre notre système totalement embarqué. L'algorithme de prédiction développé sous Matlab va donc être intégré sur Raspberry Pi. Un écran pourra être ajouté à la Pi afin d'avoir un retour d'information sur le résultat de la prédiction. Un ordinateur sera uniquement nécessaire pour la construction de la base de données et le calcul du modèle LDA, qui seront sauvegardés dans la Pi. L'interface graphique développée sous Matlab pour la base de données sera alors conservée pour faciliter sa construction.

7.1 La Raspberry Pi Zero W

La Raspberry Pi Zero W (Figure 37) a été choisie pour le développement du système embarqué. Tout d'abord en raison de son aspect très compact (65x30x5mm). De plus, elle dispose d'un Broadcom BCM2835, il s'agit d'un CPU (Central Processing Unit) à un seul cœur physique cadencé à 1GHz, avec une RAM (Random Access Memory) ou mémoire vive de 512MB (Mégabyte). Elle a également la particularité de pouvoir communiquer par Wifi ou Bluetooth. La communication Wifi sera notamment très utilisée dans ce projet.

La Pi possède plusieurs ports utiles tels que des ports micro-USB (Universal Serial Bus) (un pour l'alimentation et un autre pour la communication), un mini-HDMI (High-Definition Multimedia Interface), une prise vidéo, et des pins d'accès (headers) aux éléments matériels de la Pi. Enfin, on peut également y connecter une caméra. De plus, elle possède des ports SPI permettant une communication avec la carte d'acquisition, sans module externe.

La Raspberry Pi Zero W nécessite une carte SD (Secure Digital) d'une capacité supérieure ou égale à 8Go pour fonctionner (démarrage, écriture, lecture), d'adaptateurs tels qu'un micro-USB/USB, micro-HDMI/HDMI, d'une nappe vidéo mais aussi d'un boîtier isolant. Enfin, la Raspberry Pi étant un outil très répandu et utilisé, elle dispose d'une grande communauté riche en ressources matérielles et logicielles.

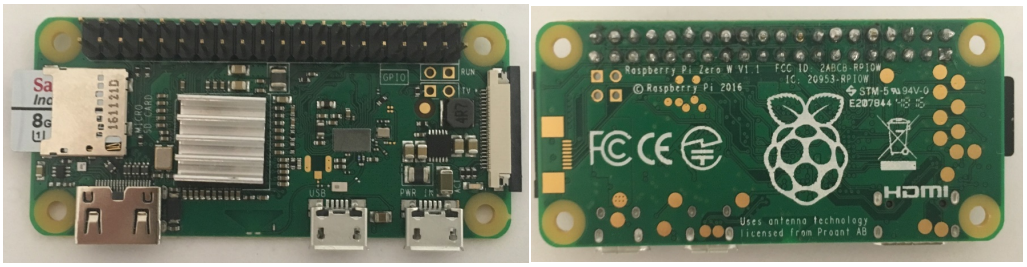

Figure 37 : Raspberry Pi Zero W.

7.2 Développement du code Matlab de classification en C++

7.2.1 Structure de l'algorithme

Tout d'abord, nous allons commencer par décrire la structure de l'algorithme développé en C++. Ce langage de programmation possède la réputation d'être rapide d'exécution et compatible sur toutes les plateformes. C'est un langage orienté objet, plutôt considéré bas niveau. Un langage objet signifie qu'on manipule des objets associés à des classes. Ainsi, nous utiliserons des classes qui rempliront des fonctions et des initialisations spécifiques. Chaque classe possède des attributs (variables propres à la classe) et des méthodes (fonctions propres à la classe), les attributs ne peuvent pas être utilisés en dehors de la classe tandis que les méthodes le peuvent. Pour utiliser les méthodes spécifiques, on doit créer (instancier) un objet du type de la classe en question.

Le code sera décomposé en plusieurs parties afin d'organiser correctement l'algorithme. Chaque classe se trouvera dans deux fichiers spécifiques (un .h et un .cpp) portant le nom de leur classe. Le .h contiendra les inclusions des fichiers utiles, les définitions de MACROS ainsi que tous les prototypes des méthodes et les déclarations d'attributs de la classe. Voici la liste les fichiers contenant des classes :

- *MySerial.cpp* et *MySerial.h* : permet la gestion de la communication UART entre la Raspberry Pi et la carte d'acquisition, ainsi que la récupération des données. Ces fichiers proviennent d'un de mes collègues Quentin Mascret.

- *Filtre_BP_BS.cpp* et *Filtre_BP_BS.h* : gère le filtrage des signaux EMG en temps réel. Une méthode réalise un filtre passe-bande entre 20 et 500Hz et un coupe-bande à 60Hz avec les mêmes coefficients que sous Matlab.

- *Tke.cpp* et *Tke.h* : calcul du TKE pour chaque canal, comme expliqué dans les parties précédentes.

- *Features.cpp* et *Features.h* : gère la totalité des calculs des paramètres discriminants pour la classification (MAV, Zero Crossing, etc.).

- *PythonBridge.cpp* et *PythonBridge.h* : permet de créer un pont entre un code Python (*LDA.py* expliqué plus loin) et notre algorithme en C++. En effet, il est possible d'utiliser un code Python (autre langage de programmation) dans du C++ en se servant d'un pont, également appelé un '*wrapper*'. Ces fichiers proviennent de mes collègues Ulysse Côté-Allard et Quentin Mascret, et sont adaptés à mon code.

- *classifiercollector.cpp* et *classifiercollector.h* : cette classe utilise la classe précédente pour travailler. Elle crée des méthodes permettant d'utiliser simplement les méthodes Python de *LDA.py*, notamment pour la classification. Ces fichiers proviennent d'un de mes collègues Quentin Mascret, et sont adaptés à mon code.

Les fonctions particulières, non associées à des classes, seront regroupées dans un fichier *functions.cpp* avec leurs prototypes dans le fichier d'en-tête *functions.h*. En plus de cela, un fichier d'en-tête *types.h* sera rajouté afin de définir des MACROS telles que la valeur des touches clavier ou des couleurs d'affichages sur le terminal de la Pi.

Un fichier *LDA.py* contient une classe codée en Python pour l'application de la méthode LDA. La librairie *scikit-learn (sklearn)* et notamment l'entité *LinearDiscriminantAnalysis* sont importées ici. Grâce à cela, nous pouvons alors entraîner le système avec notre base de données et générer un modèle basé sur ces résultats d'entraînement. De plus, cette librairie offre la possibilité d'effectuer des prédictions. Nous avons choisi de l'utiliser, car elle se rapproche beaucoup de la méthode LDA sous Matlab (Eigenvalue).

Ainsi, nous nous servirons de l'interface graphique Matlab pour construire la base de données, puis nous enregistrerons celle-ci dans un dossier spécifique, placé dans le répertoire de l'algorithme. Dans ce dossier, on trouvera 2 fichiers, le premier étant la matrice des paramètres et le deuxième le vecteur des classes : *matrix_features.txt* et *vector_classes.txt*. Après ces manipulations, l'algorithme sera en mesure de construire le modèle linéaire (enregistré dans le fichier *LDA.pkl*) automatiquement au démarrage.

Nous pouvons inscrire le nom du dossier, contenant la base de données, à la dernière ligne du fichier *LDA.py*, si nous souhaitons exécuter ce fichier indépendamment du reste (commande : *python LDA.py*). Cela nous permet notamment de construire le modèle LDA sur un ordinateur de travail plus puissant que la Raspberry Pi Zero afin d'accélérer la création du modèle. Une fois que celui-ci est créé pour une base de données spécifique, l'algorithme ne le recréera pas et donc il démarrera plus rapidement.

Pour terminer sur la structure du code, le fichier *main.cpp* fait le lien entre tous les fichiers précédents. C'est lui qui appellera l'ensemble des constructeurs associés à chaque classe ainsi que les méthodes sous-jacentes. Deux processus synchronisés ('*thread*') sont à l'œuvre dans ce fichier. Un processus permettra la récupération et le triage des données reçues par UART. Tandis que le deuxième processus s'occupera du traitement des données et de la classification. La méthode multifilaire va permettre d'optimiser les temps de calcul pour ainsi prédire en temps réel de façon fluide.

7.2.2 Librairies utilisées

Les librairies externes incluses dans notre code peuvent être en C ou en C++, car il est possible d'utiliser un langage C dans du C++. Nous allons ici les passer en revue :

Librairies C et C++ :

- Pour la gestion des allocations, librairie standard : *stdlib.h* (C).
- Pour les entrées et sorties de textes à l'écran : *cstdio* (C++), *stdio.h* (C), *iostream* (C++).
- Pour l'utilisation des chaînes de caractères : *cstring*, *string* (C++/C).
- Pour les fonctions mathématiques complexes : *cmath* (C++).
- Pour lire et écrire dans un fichier (gestion des flux) : *fstream* (C++).

- Pour lire le contenu d'un dossier : *dirent.h* (C).
- Pour mettre en place des processus parallèles : *pthread.h* (C).
- Pour la gestion du temps : *ctime* (C++).
- Pour la gestion noyau des évènements USB (fixer le Baud Rate…) : *termios.h* (C), *unistd.h* (C), *asm/termbits.h* (C).
- Pour le contrôle des fichiers : *fcntl.h* (C), *sys/ioctl.h* (C).
- Pour avoir des informations à l'écran sur les erreurs : *errno.h* (C).
- Pour les transformations sur des vecteurs (rotations, permutations…) : *algorithm* (C++).
- Pour la gestion des tableaux en C++ : *vector* (C++).
- Pour travailler avec des complexes : *complex.h* (C).

- Pour le calcul de la Transformée de Fourier, nous utiliserons la librairie *fftw3.h* (C), car elle est utilisée sous Matlab.
- Librairie pour '*wrapper*' du python (v 2.7) en C++ : *python2.7/Python.h* (C).

Librairies Python :

- *Scikit-learn (sklearn)* est une librairie de référence dans le monde de l'intelligence artificielle et des modèles d'apprentissages dans le langage Python. Elle est très connue pour sa simplicité et sa rapidité d'exécution. On utilise ici : *sklearn.discriminant_analysis/LinearDiscriminantAnalysis* (utilisation du modèle LDA), et *sklearn.externals/joblib* (charger et sauvegarder le modèle).
- Pour la gestion des divisions euclidiennes sous Python : *__future__/division*.
- Pour la gestion des vecteurs sous Python : *numpy*.
- Pour la gestion des fichiers sous Python : *os*.

7.2.3 Description du fichier principal

Afin d'alléger les explications, seul le fichier principal sera analysé ici. Les codes au complet pourront être retrouvés dans l'Annexe C.

Ainsi, le *main.cpp* permet de faire le lien entre tous les fichiers créés. Nous définissons un ensemble de variables globales au début de celui-ci. Ces variables pourront être utilisées dans l'ensemble de l'algorithme. De même, on définit une structure qui sera utilisée pour la lecture et l'écriture des données (dans les deux processus), puis on alloue dynamiquement des pointeurs pour manipuler un grand nombre de valeurs. De plus, on instancie des objets associés aux classes utiles et on prépare l'algorithme à recevoir un évènement clavier (appui sur une touche).

Il est très important de comprendre que l'algorithme va se dérouler en plusieurs étapes. La première est de faire les initialisations précédentes, puis d'entrer dans le *main*. Dans celui-ci, nous allons initialiser les variables qui n'ont pas pu l'être précédemment et allouer dynamiquement l'ensemble des pointeurs. Après cela, nous chargerons la base de données, et configurerons le port USB sur lequel viendra se brancher la carte d'acquisition. Pour terminer, les *threads* seront configurés et lancés lors de l'appui sur la touche 'D' du clavier. Les *threads* sont en fait deux processus parallèles qui partagent les mêmes espaces mémoires.

Ils permettent d'exécuter des tâches en parallèle. Ici, ces processus seront verrouillés par des *mutex* afin de les rendre synchronisés. En d'autres termes, lorsqu'un processus est verrouillé, seul le processus qui a verrouillé le *mutex* peut lire et écrire les variables partagées, l'autre doit attendre qu'il soit déverrouillé. Ces processus sont le cœur de cet algorithme, car ils vont nous permettre de synchroniser la réception des données et les traitements/calculs associés à ces données. Les *threads* sont donc lancés à ce niveau de l'algorithme.

Le premier processus est nommé *read_data* tandis que le deuxième se nomme *treatment_data*. Des variables locales sont initialisées et des pointeurs locaux sont alloués dynamiquement au début de chacun de ces processus. Dans *read_data*, le port UART est ouvert. Puis une boucle, pour chaque processus, se lance avec une condition d'arrêt des *threads* : les processus peuvent être arrêtés définitivement lors de l'appui sur la touche 'E' du clavier.

Tout d'abord, nous allons voir ce qui se déroule dans *read_data*. Ce processus permet la réception des données par UART et leurs triages. En premier lieu, nous devons faire l'acquisition des données par UART. Deux cas peuvent se présenter au début de cette routine, soit c'est la première fois que l'on passe dans la routine, soit l'acquisition a déjà été lancée et ce n'est pas la première fois que la boucle se fait. Dans le deuxième cas, on récupère simplement un paquet de taille 450 par UART. Dans le premier cas, par contre, nous devons synchroniser la lecture du buffer de la Pi avec l'envoi des paquets par le microcontrôleur. Pour ce faire, on lit les données une par une jusqu'à trouver le bit de début du paquet. Une fois trouvé, on lit le reste du paquet et on vérifie que le bit de fin se trouve à la bonne place. Si c'est le cas, alors la synchronisation est réussie et on peut passer à la suite.

Nous avons donc à notre disposition un tableau de caractères, qu'on transforme en tableau d'entiers, contenant les 450 données (avec les bits de début et de fin) envoyées par le microcontrôleur. Ces données ne sont pas correctement triées par canal, nous allons donc le faire et regrouper les deux moitiés d'échantillons en un grâce au complément à deux. Cela va nous permettre d'obtenir une matrice avec chacune des lignes correspondantes à un canal (7 lignes), et chaque colonne contenant un échantillon (32 échantillons/canal pour un paquet de 450 données).

On peut alors remplir le buffer partagé entre les deux processus avec ces 32 premières valeurs/canal. À ce niveau, on bloque l'accès aux données communes (structure *mybuffer_data_t*) pour que le deuxième processus (*treatment_data*) ne modifie pas les variables et buffer commun. On incrémente la tête (*head*) de ce buffer afin que les prochaines valeurs lues par UART se retrouvent derrière les premières. Le buffer partagé peut être de taille allant jusque 960 données, afin de ne pas avoir de pertes de données si la lecture de l'UART est plus rapide que les traitements. Après cela, on déverrouille les données communes et on informe le deuxième processus que des données ont été rajoutées dans le buffer partagé, pour que celui-ci se réveille.

Enfin, dans *read_data*, on s'occupe également de la gestion du clavier. En cas d'appui sur 'E', on sort des threads et on termine le programme. En cas d'appui sur 'Q', on effectue la calibration pendant 5s et on affiche les seuils de chaque canal. Cela conclut les explications sur le premier processus, *read_data*.

Nous allons maintenant décrire le déroulement des opérations dans le second processus, *treatment_data*. On initialise les variables qui doivent être réinitialisées à chaque

début de la routine, puis on verrouille les données communes pour empêcher l'autre thread (*read_data*) de les modifier. On regarde alors dans le buffer partagé si 192 nouvelles données/canal sont disponibles. Si c'est le cas, on met ces données, remises à l'échelle par rapport à la plage de conversion de l'ADS, dans une nouvelle matrice *output*. Puis on incrémente la queue (*tail*) du buffer commun afin de ne plus utiliser ces données la prochaine fois. Enfin, on déverrouille les données communes. Si par contre, les 192 données/canal ne sont pas encore disponibles, on se met en sommeil en déverrouillant les données communes. Lorsque *read_data* nous informera que de nouvelles données sont présentes, alors on effectuera une vérification jusqu'à avoir 192 données de disponibles.

Ainsi, nous avons à notre disposition une matrice *output* contenant 192 données/canal. Nous pouvons maintenant commencer les traitements. D'abord, un filtrage passe-bande entre 20 et 500Hz et un coupe-bande à 60Hz (mêmes coefficients que sous Matlab) sont mis en place. On utilise pour cela des objets de la classe *Filtre_BP_BS* qui nous permettent de filtrer chaque canal. Puis on calcule l'opérateur TKE pour chaque canal (résultats dans *output_TKE*), avec un objet de la classe *Tke*.

On arrive au moment du calcul des paramètres discriminants et de la prédiction. Si la calibration est demandée par 'Q', alors on calcule le seuil de chaque canal qui est le maximum des valeurs TKE prises sur 5s avec la main ouverte (mouvement neutre). Si la calibration n'est pas demandée, alors soit elle a déjà été faite, soit non et dans ce cas les seuils sont tous à 0. On détecte la présence d'un mouvement (« onset detection ») lorsqu'au moins une valeur d'un des canaux dépasse son seuil. Dans le cas contraire, on estime être dans le cas d'un mouvement neutre et on ne calcule pas les paramètres discriminants. Si la calibration n'a pas été faite, les seuils étant tous à 0, on effectue le calcul des paramètres discriminants et la prédiction. Cette calibration peut être très utile pour éviter certains calculs inutiles, et cela permet de renforcer l'efficacité de prédiction du mouvement neutre. Il est tout à fait possible qu'on soit dans le cas d'un mouvement neutre, mais qu'on fasse la prédiction, car on détecte une activité. Cela n'induit pas d'erreur, car le neutre fait partie d'un des mouvements à prédire et donc la prédiction renverra main ouverte.

Donc, si une activité musculaire est repérée, on calcule les paramètres discriminants en utilisant un objet de la classe *Features*. On obtient alors un vecteur contenant l'ensemble des paramètres calculés (*features_vector*). Ce vecteur est envoyé au classifieur sous Python via le *wrapper (PythonBridge)*, c'est un objet de la classe *classifiercollector* qui s'occupe de ces manipulations. La variable *winning_finger* contient alors le numéro du mouvement prédit. On affiche le mouvement prédit sur le terminal de la Pi. Cela conclut les opérations effectuées pour le processus *treatment_data*.

À la fermeture du programme, après l'appui sur 'E', toutes les variables allouées dynamiquement dans les processus sont désallouées afin de libérer la mémoire. Puis les threads sont fermés. Enfin, on retourne dans le fichier principal afin de désallouer toutes les variables globales allouées dynamiquement.

7.3 Portage du code sur Raspberry Pi Zero

Après l'écriture et le débogage de l'ensemble de l'algorithme, le code a été porté et débogué sur Raspberry Pi Zero grâce à des protocoles de communication sécurisés (*ssh* WIFI

ou USB). Une carte SD de 8Go sur la Pi est utilisée pour la sauvegarde du code. Le système d'exploitation (OS : Operating System) de la Pi est Raspbian et son noyau UNIX.

Une fois le code importé, il faut compiler puis exécuter l'algorithme dans le projet sur la Pi. Afin de rendre la compilation de l'ensemble des fichiers plus simple, un '*Makefile'* a été créé. Pour cela, on écrit un *CMakeLists.txt* qui contient des informations sur l'ensemble des fichiers à compiler. Dans le terminal de la Pi, la commande *cmake* est utilisée afin de créer le *Makefile*. Une fois celui-ci créé, nous pouvons utiliser la commande *make*. Cela permet alors de compiler chaque fichier indépendamment, créer un fichier binaire pour chaque fichier puis finalement faire le lien entre tous ces fichiers binaires pour fournir un exécutable commun. Lorsque toutes ces opérations sont réalisées, l'exécutable peut être lancé avec la commande : *./bin/Project_EMG*.

Après le lancement de l'algorithme, les différentes étapes sont affichées sur le terminal de la Pi afin de guider l'utilisateur : chargement de la base de données, ouverture de l'UART, démarrage, calibration, prédiction, etc. Le terminal de la Pi peut être affiché sur un écran connecté à la Pi ou sur un écran à distance configurable sur un ordinateur. Les touches 'D' (démarrer), 'E' (fermer le programme), et 'Q' (calibration) d'un clavier permettent de contrôler le déroulement de l'algorithme.

7.4 Expérimentations sur la Raspberry Pi Zero

Nous avons testé le déroulement de cet algorithme sur Raspberry Pi Zero. Les tests ont été réalisés sur un participant afin de confirmer la validité de fonctionnement de l'algorithme. Le participant a dû construire sa base de données avant de vérifier les prédictions de chacun des mouvements en temps réel ainsi que la fluidité du système. Le terminal de la Pi a été affiché sur notre ordinateur de travail (communication *ssh*) afin d'observer les résultats (voir Figure 38). Une batterie de 20Ah (Aukey) est utilisée pour alimenter la Pi, la capacité de celle-ci pourrait, sans difficulté, être réduite. En effet, la Raspberry Pi consomme environ 0,85W (170mA à 5V) donc celle-ci pourrait fonctionner pendant environ 5jours avec la batterie choisie.

Ainsi, la base de données a été construite sous Matlab puis exportée sur la Pi. L'efficacité a été calculée pour une base de données spécifique et elle est de 94%. Nous avons pu constater une certaine fluidité de la prédiction malgré quelques ralentissements au démarrage de l'algorithme et à certains moments. Il faut attendre environ 30s avant d'obtenir une très bonne fluidité de prédiction. Cela est essentiellement dû aux performances limitées de la Raspberry Pi Zero. Nous travaillons à la limite de ses capacités et son CPU est parfois saturé à 100%. Nous pourrions augmenter sa rapidité de travail en changeant sa fréquence d'horloge. Ou nous pourrions passer sur une Raspberry Pi 3 beaucoup plus puissante : le code a été testé sur celle-ci et aucun retard n'a pu être observé. Les temps d'exécution du code sont d'environ 180ms avec la Pi Zero, et 110ms avec la Pi 3.

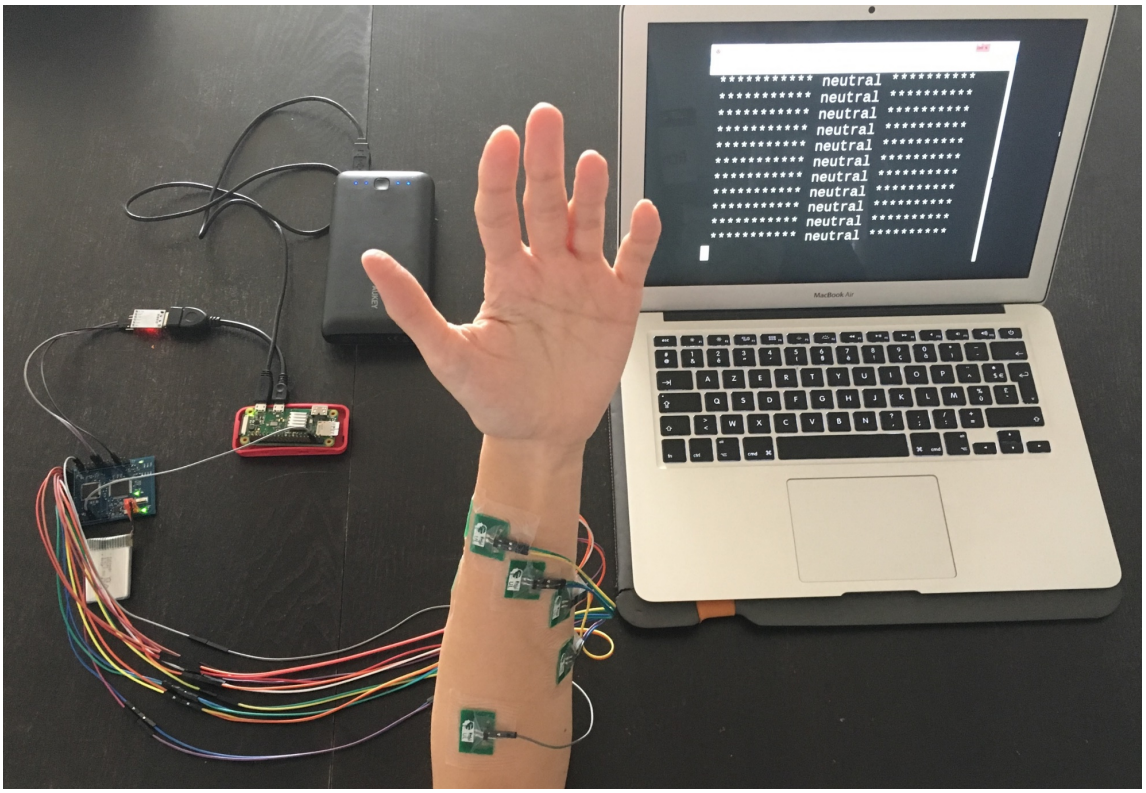

Figure 38 : Vue générale du système.

Au niveau des résultats de prédiction, on a pu observer que tous les mouvements étaient très bien prédits, excepté la flexion du pouce. Ce mouvement est confondu avec la prise cylindrique, car la contraction du pouce est également nécessaire pour la prise cylindrique. Nous pourrions remédier à cela en augmentant le nombre d'observations des différents mouvements dans la base de données. Ces prédictions permettront de décider des tensions analogiques de sortie sur 3 des pins de la Raspberry Pi connectées à la consigne des moteurs de la prothèse (voir la partie suivante).

Des écrans tactiles spécifiques pour Raspberry Pi existent et pourraient être une bonne solution pour visualiser les résultats de prédiction tout en ayant la possibilité de se servir des touches de contrôle.

8 Design de la prothèse mécatronique

Dans cette partie, nous allons aborder le design de la prothèse mécatronique. Le système de détection des mouvements de la main pourra être utilisé avec cette prothèse afin de commander les moteurs.

8.1 Main en impression 3D

Pour ce projet de recherche, nous avions à notre disposition une main en impression 3D (3 dimensions) possédant 5 doigts articulés via des câbles (Figure 39). Cette main provient du laboratoire de robotique de l'Université Laval [70]. Nous avons alors décidé de nous servir de cette main et de la motoriser. Pour simplifier le contrôle, nous avons utilisé un moteur afin d'activer le mouvement du pouce, un autre moteur pour l'index et le majeur combiné, tandis qu'un dernier moteur nous permettra de motoriser l'annulaire et l'auriculaire. Il a été nécessaire de choisir d'utiliser 3 moteurs plutôt qu'un seul, car nous avons besoin de réducteurs en sortie. Or le problème des réducteurs est qu'ils ne peuvent pas supporter un grand couple en sortie (0,27N.m pour les réducteurs choisis). Donc utiliser un unique moteur ne nous aurait pas permis d'obtenir un couple suffisant pour fermer la main. De plus, ceci offre plus de flexibilité en termes de prises.

Nous avons pu déterminer certains besoins quant à la force de serrage de la main et la vitesse de déplacement des doigts. Un câble en sortie permet de fermer les doigts. Nous avons déterminé que pour une prise relativement robuste, la force de tension nécessaire sur ce câble était de 100 Newtons (à vitesse nulle). Pour fermer les doigts à une vitesse de 10cm/s, une force de 25N est requise. Ainsi, chaque moteur devra fournir une force de 33N à vitesse nulle et 8,33N à 10cm/s.

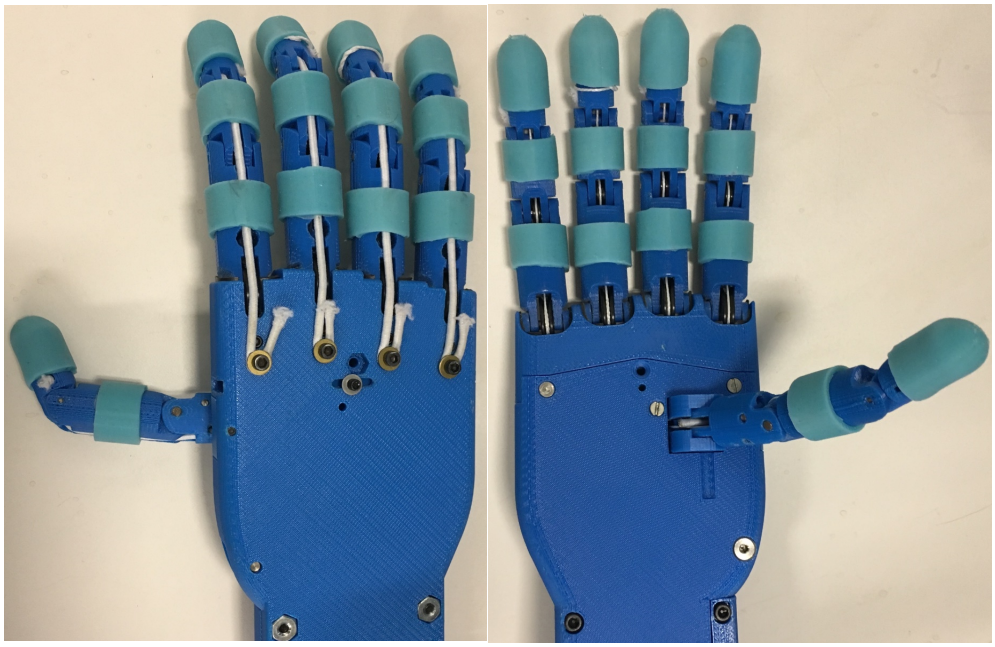

Figure 39 : Main en impression 3D.

8.2 Choix des moteurs, réducteurs et encodeurs

Afin de satisfaire aux besoins déterminés dans la partie précédente, nous nous sommes penchés sur le choix des moteurs ainsi que des réducteurs. Les réducteurs vont notamment pouvoir amplifier le couple en sortie du moteur, afin d'obtenir des forces plus importantes. De plus, nous aurons besoin de choisir des encodeurs afin d'avoir un retour d'information sur l'état des moteurs.

8.2.1 Type de moteur

La question du type de moteur s'est d'abord posée. En effet, beaucoup de types différents existent : Moteur DC, Moteur Brushless DC, Servomoteur rotatif, Moteur Piézoélectrique, etc. Une comparaison des performances des principaux moteurs pouvant être utilisés pour notre prothèse a été réalisée (Tableau 6).

Tableau 6 : Avantages et Inconvénients de différents types de moteurs.

Types de moteurs	Avantages	Inconvénients
Moteur DC	- Large plage de variation de vitesse (si accompagné d'un variateur de vitesse) - Vitesse de rotation élevée - Régulation précise du couple - Indépendant de la fréquence du réseau - Faible coût - Simplicité de commande - Miniaturisation possible	- Usure rapide à vitesse élevée (dû aux collecteurs et aux balais) - Faible couple - Nécessité d'un réducteur
Moteur Brushless DC	- Grande durée de vie et bonne fiabilité - Faible encombrement et poids (2 à 3 fois plus léger qu'un moteur DC) - Rendement bien supérieur au moteur DC (moins de frottement sans balais) et ce sur toute la gamme de vitesse - Vitesse importante - Excellent rapport poids/puissance - Rapide au démarrage - Accélération élevée grâce à une bonne commutation - Faibles bruits et vibrations - Bonne variation de vitesse avec maintien du couple	- Plus coûteux - Nécessité d'un réducteur - Usure rapide à vitesse élevée
Servomoteur Rotatif	- Atteint et maintient une position angulaire fixe (bonne précision) - Vitesse de rotation élevée - Mécanique et électronique intégrées dans un seul boîtier	- Puissance faible - Fragile - Souvent coûteux - Consommation importante - Encombrement moyen

Moteur Piézoélectrique	- Très bon rapport poids/puissance - Pas besoin de réducteur : couple élevé - Faible encombrement et poids - Niveau de bruit très faible voir nul (vibrations ultrasonores) - Très réactif (1ms au lieu de 100ms en général) - Très précis - Ne consomme pas d'énergie à l'arrêt (pression du rotor sur le stator bloque le mouvement naturellement) - Pas de risque de perturbations électromagnétiques - Design très flexible : miniaturisation facile	- Faible durée de vie (dû à l'usure par friction) - Coût élevé - Alimentation électrique complexe (concordance des 2 voies) - Faible rendement (10-25%), car entraînement par friction de la céramique piézo - Besoin d'une source électrique haute fréquence

Malgré un très bon contrôle de la position, le servomoteur semble être un choix moins judicieux du fait de son encombrement moyen et de sa consommation importante. Les moteurs Piézoélectrique et Brushless DC ont tous les deux l'avantage d'avoir une vitesse de rotation élevée, un faible encombrement, des niveaux de bruit relativement faibles, de bons rapports poids/puissance et une bonne réactivité. Par contre, le moteur Brushless possède un bien meilleur rendement que le moteur Piézoélectrique et une grande durée de vie. Tandis que le moteur Piézoélectrique a un couple important, une très bonne précision et aucun risque de perturbations électromagnétiques. Le problème majeur de ces moteurs est le coût élevé de ceux-ci. Or, nous souhaitons faire une prothèse accessible à tous et donc que le prix soit raisonnable. C'est pourquoi ces moteurs ne seront pas retenus.

Ainsi, notre choix s'est porté sur le moteur DC. Le moteur à courant continu est celui le plus utilisé actuellement dans les prothèses myoélectriques, car il est peu coûteux, possède de bonnes performances et est facile à miniaturiser. Il est constitué d'un stator (partie fixe composée d'aimants permanents) et d'un rotor qui est la partie mobile composée de bobines. Le courant électrique, traversant les bobines, crée un champ magnétique qui s'oppose vectoriellement au champ magnétique de l'aimant permanent. Le rotor est placé sur un axe qui entre alors en rotation. L'ensemble collecteur-balais assure mécaniquement la commutation dans l'alimentation des bobines en fonction de l'angle du rotor afin d'entretenir le mouvement. Le fonctionnement peut être moteur ou générateur.

Nous avons rajouté un réducteur au moteur DC afin d'augmenter le couple en sortie. Il permet de modifier le rapport vitesse/couple entre l'axe d'entrée et de sortie du mécanisme. Ses avantages sont d'être économique, peu encombrant, et de supporter une grande charge.

Un encodeur incrémental de bonne résolution a été ajouté à la configuration des motoréducteurs afin d'avoir une information de retour sur la vitesse ou la position.

Ainsi après des recherches, nous avons choisi l'entreprise Maxon pour commander nos moteurs. Voici la configuration des moteurs/réducteurs/encodeurs (Figure 40) :

- Moteur : DCX14L GB KL 6V
- Réducteur : GPX14 C 28:1
- Encodeur : ENX10 EASY 256IMP

Dans un premier temps, un moteur équipé d'un réducteur a été testé, puis deux motoréducteurs montés avec encodeurs. La partie suivante présente les calculs effectués pour la vérification des besoins, quant à la force de serrage de la main et la vitesse de déplacement des doigts.

(a) (b) (c)

Figure 40 : **(a)** Moteur, **(b)** Réducteur et **(c)** Encodeur incrémental.

8.2.2 Vérification des besoins

Notre but est maintenant de vérifier les bonnes dimensions des motoréducteurs, ainsi que le couple et la consommation d'énergie. La Figure 41 présente le schéma bloc de notre configuration de motoréducteur ainsi que les formules importantes.

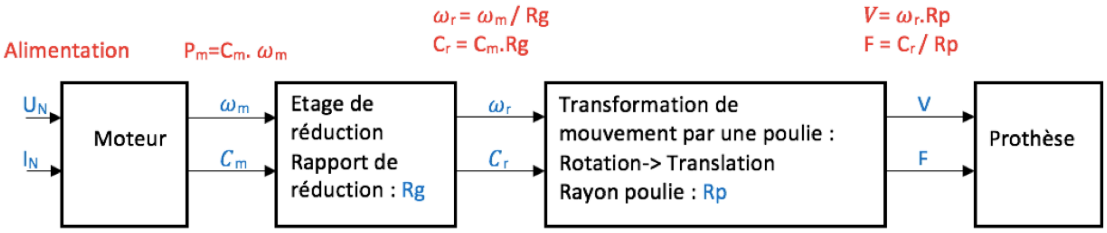

Figure 41 : Configuration d'un motoréducteur.

Tout d'abord, nous avons choisi le plus petit rapport de réduction (Rg) possible pour le réducteur, afin de réduire son couple en sortie et éviter d'atteindre ses limites. Pour compenser, on prendra un rayon de poulie très petit (Rp = 6mm). La poulie permet de transformer le mouvement de rotation du moteur en un mouvement de translation des câbles activant les doigts de la prothèse.

Même si nous voulons miniaturiser au maximum la prothèse, les moteurs ne doivent pas être trop petits, car sinon ils ne seront pas assez puissants. C'est le moteur Maxon DCX14L GB KL 6V, qui a été retenu. Voici ses caractéristiques :

- Diamètre : 14mm
- Longueur : environ 45mm
- Poids : 26g
- Puissance mécanique : 5,38W
- Puissance d'alimentation : 7,86W
- Tension nominale : 6V
- Courant nominal : 1,31A

- Couple nominal : 6,92mN.m
- Vitesse nominale : 7430rpm ou 778rad/s
- Rendement : 80,4%
- Roulements à billes
- Brosses en Graphite : les brosses permettent de conduire le courant entre la partie fixe et mobile du moteur. Différents matériaux peuvent être utilisés comme du graphite (rigide), du carbone ou du cuivre pour une meilleure conductivité. Ces brosses en Graphite sont bien adaptées pour un fort courant continu et des applications avec des pics de courant.

Le réducteur GPX14 C 28:1 a été retenu. Voici ses caractéristiques :
- Diamètre : 14mm
- Longueur : 32mm
- Rapport de réduction : 28:1
- Version Céramique : car peut supporter un couple de sortie plus important.
- 2 étages de réduction
- Rendement : 80%

Pour les calculs nous utiliserons les formules :

$$F\,(N) = \frac{Cm.Rg}{Rp} \tag{12}$$

$$V\,(m/s) = \frac{\omega m.Rp}{Rg} \tag{13}$$

$$1\ rpm = \frac{2\pi}{60}\ rad/s \tag{14}$$

Avec Cm, le couple moteur nominal, Rg le rapport de réduction du réducteur, Rp le rayon de poulie, ωm la vitesse nominale du moteur en rad/s.

Au maximum, on demandera une force de 33N au moteur, c'est pourquoi nous avons choisi Rg = 28.

$$\frac{Cm.Rg}{Rp} = \frac{0{,}00692 \times 28}{0{,}006} = 32{,}3N = F \tag{15}$$

Il est également très important que la condition suivante soit satisfaite afin d'éviter d'endommager le système :

$$C_{N.}R_g.\,rendement\_reducteur \leq C_{sortie\_max\ \_reducteur} \tag{16}$$

Or, $C_{N.}R_g.\,rendement_{reducteur} = 0{,}155N.\,m$
Et, $C_{sortie\_max\ \_reducteur} = 0{,}27N.\,m$

Donc, la condition (12) est bien vérifiée : 0,155 < 0.27.

Ainsi les motoréducteurs correspondent à nos besoins pour la prothèse mécatronique.

8.3 Choix des contrôleurs

Après le choix des moteurs/réducteurs/encodeurs, il a fallu choisir des contrôleurs adaptés. Nous sommes restés sur la gamme Maxon, avec le ESCON Module 24/2 (Figure 42). Ce contrôleur permet le contrôle d'un moteur DC.

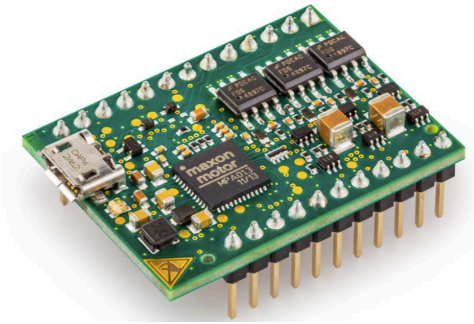

Figure 42 : ESCON Module 24/2 de chez Maxon.

Cette carte doit être alimentée entre 10 et 24V, car elle alimente elle-même le moteur connecté à ses bornes. On la programme grâce à l'interface graphique développée par Maxon : ESCON Studio (Figure 43). Il suffit alors de rentrer toutes les configurations du motoréducteur utilisé ainsi que de son encodeur, et de définir le mode de commande. On se place en mode régulateur de vitesse. Ce mode est possible grâce au retour d'information de vitesse mesurée par l'encodeur incrémental, lui-même branché aux pins du contrôleur qui lui sont dédiées. Puis, on applique une tension continue sur une des entrées analogiques. C'est en faisant varier la valeur de cette tension que la vitesse varie proportionnellement. On peut choisir la plage de variation de vitesse et celle en tension tout en respectant les valeurs limites du moteur. Nous pouvons, par exemple, tester une vitesse entre -15 000rpm et +15 000rpm correspondante à une tension de 0V à 10V, 5V étant la vitesse nulle. Ce sont ces tensions analogiques qui seront fournies par la Raspberry Pi Zero (à l'aide de ses sorties numériques PWM en 0/3,3V) afin de mettre les moteurs en marche et faire correspondre la position de la prothèse au mouvement prédit.

Une des fiches d'entrées numériques du contrôleur permet le blocage rapide du moteur en cas de problème : elle doit être en tout temps alimentée à 3,3V pour que le moteur fonctionne. On peut également configurer des fiches de sorties analogiques afin d'avoir des informations sur le moteur : par exemple, le courant moteur réel ou la vitesse réelle. Les tensions de sortie sont alors proportionnelles aux informations voulues.

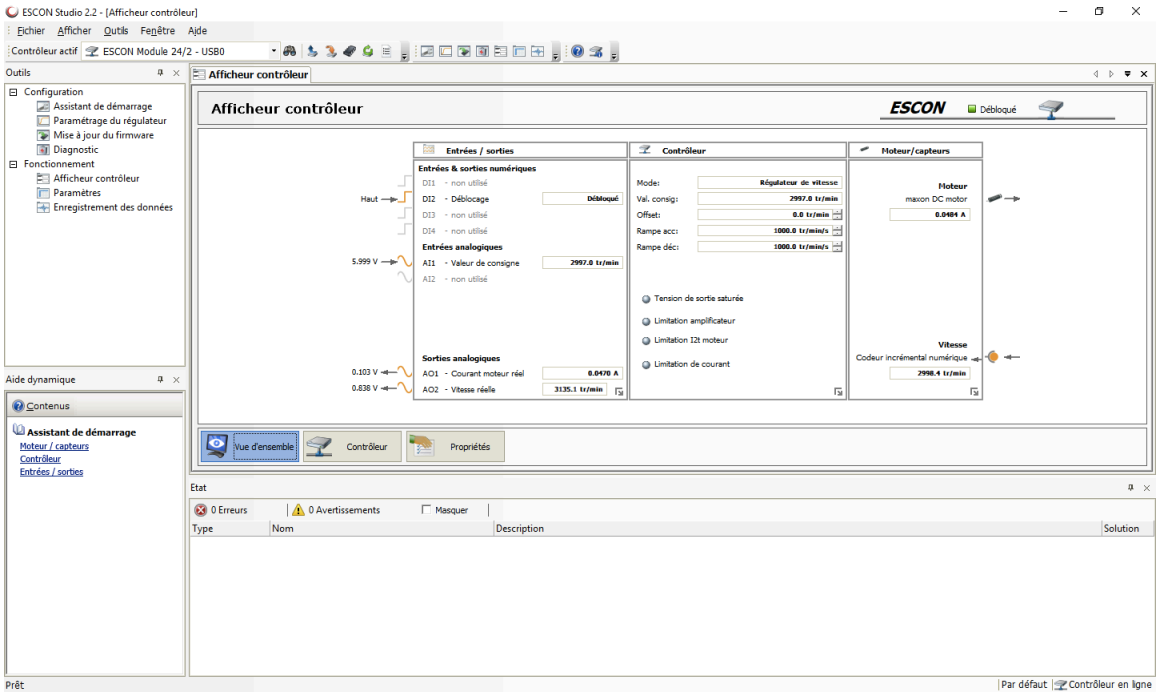

Figure 43 : Interface graphique ESCON Studio de chez Maxon, pour le contrôleur.

Ainsi, nous aurons besoin de 3 contrôleurs ESCON Module 24/2 pour contrôler nos 3 motoréducteurs et nous assurer du bon fonctionnement de ceux-ci.

8.4 Design final

Nous avons donc pu réunir tous les composants nécessaires à la prothèse mécatronique : les moteurs, les réducteurs, les encodeurs, les contrôleurs, la main en impression 3D et les systèmes de transformation de mouvement par poulies. Il reste à assembler l'ensemble afin que les contrôleurs activent les motoréducteurs qui, reliés aux poulies, permettent la translation des câbles rattachés aux doigts de la main.

Pour cette partie, nous avons fait appel à un professionnel de recherche, Simon Foucault, pour le design final de la prothèse. Voici le rendu final de celle-ci :

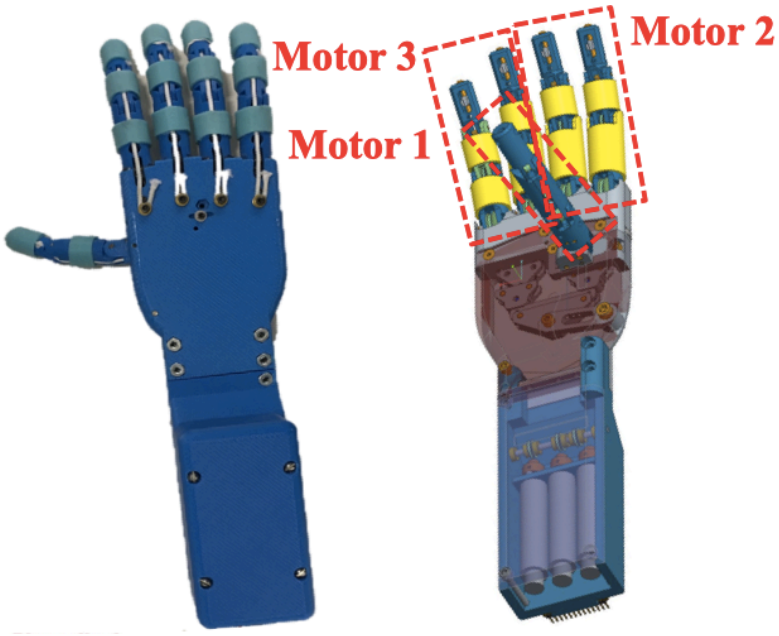

Figure 44 : Design final de la prothèse mécatronique.

Il est possible de calculer le prix final de la prothèse et du système de contrôle :

Tableau 7 : Prix du système développé.

Matériel	**Prix ($CAD)**
Ensemble motoréducteurs/encodeurs/contrôleurs	2 085
Kit Raspberry Pi Zero W	55
Carte d'acquisition et composants électroniques	95
Électrodes	40
Batteries	20
Fils et connecteurs	20
Main 3D, poulies 3D, boîtier 3D	400
Prix final approximatif ($CAD)	2 700

Au total, le prix final est d'environ 2 700 $CAD. Ce qui est bien inférieur au prix des prothèses commerciales courantes (Tableau 1). La différence de prix se trouvant essentiellement dans le choix des matériaux de la prothèse, imprimés à l'aide d'une imprimante 3D, et au niveau de la conception des électrodes (habituellement bien plus coûteuses).

9 Expérimentation

Dans cette partie, des tests seront réalisés sur 8 participants. Le but étant de pouvoir calculer les efficacités de prédiction de notre algorithme de classification des mouvements. De plus, nous pourrons analyser la qualité des signaux EMG enregistrés ainsi que la fluidité de la prédiction en temps réel.

9.1 Protocole expérimental

La Figure 45 représente le système de détection en temps réel au complet.

Figure 45 : Système de détection en temps réel au complet et interface graphique utilisée pour la construction de la *database*.

Dans ce mémoire, nous utiliserons donc 7 électrodes sèches, ainsi qu'une électrode de référence, placées sur le bras comme le montre la Figure 20. Les 13 mouvements de la main à reconnaître sont présentés sur la Figure 32. Les tests seront effectués sur 8 sujets en santé, 2 hommes et 6 femmes, de conditions physiques différentes et avec un âge variant de 17 à 77 ans. Un des participants est sujet à des tremblements importants de la main, d'origine neurologique. Pour chacun de ces sujets, une base de données va être enregistrée de la manière décrite dans la partie précédente. Puis, la détection en temps réel pourra être testée afin de pouvoir juger de la qualité des prédictions.

Pour l'enregistrement des mouvements, nous avons demandé aux sujets d'effectuer les 13 mouvements le plus naturellement possible avec de légères variations d'intensité et de vitesse afin de rendre la *database* robuste. Une période de familiarisation de 5min environ a été donnée aux sujets afin de leur expliquer le protocole et leur montrer les signaux EMG en temps réel. Chacun des 13 mouvements seront reproduits 10 fois de suite pendant 5s. Cela demandera une certaine concentration, car les mouvements enregistrés dans la base de données devront pouvoir être reproduits en temps réel afin que le modèle puisse prédire

correctement les mouvements. C'est pourquoi il est important d'enregistrer, dans la *database*, les différentes variations possibles de mouvement. En moyenne, la pose des électrodes sur les muscles spécifiques prendra 30min, puis il faudra 1h d'enregistrement pour la base de données. Puis, 5min pour tester la prédiction en temps réel.

Nous calculerons les efficacités de prédiction pour chacun des mouvements et l'efficacité totale.

9.2 Efficacité

Dans cette partie, nous allons analyser les résultats obtenus. Tout d'abord, la Figure 46 nous permet de constater l'efficacité du filtrage et la qualité d'acquisition des signaux EMG. Le 60Hz provenant de l'interférence avec les alimentations a été supprimé à l'aide d'un filtre coupe-bande IIR d'ordre 2 (vu dans la partie 5). De même, les fréquences au-dessus de 500Hz et en-dessous de 20Hz sont atténuées grâce au filtre passe-bande IIR d'ordre 2 (vu dans la partie 5 [60]), ce qui nous permet d'avoir accès à la plage utile des signaux EMG. La Figure 46 (c) présente la Transformée de Fourier du signal EMG (46 (a)).

La forme TKE de la Figure 46 (b) nous permet notamment d'identifier très clairement la partie utile du mouvement. Rappelons que le TKE a été utilisé pour extraire 5s de mouvement dans des enregistrements de 8s pour la construction de la base de données. Et il est utilisé en temps réel afin de déterminer la présence d'un mouvement ou non.

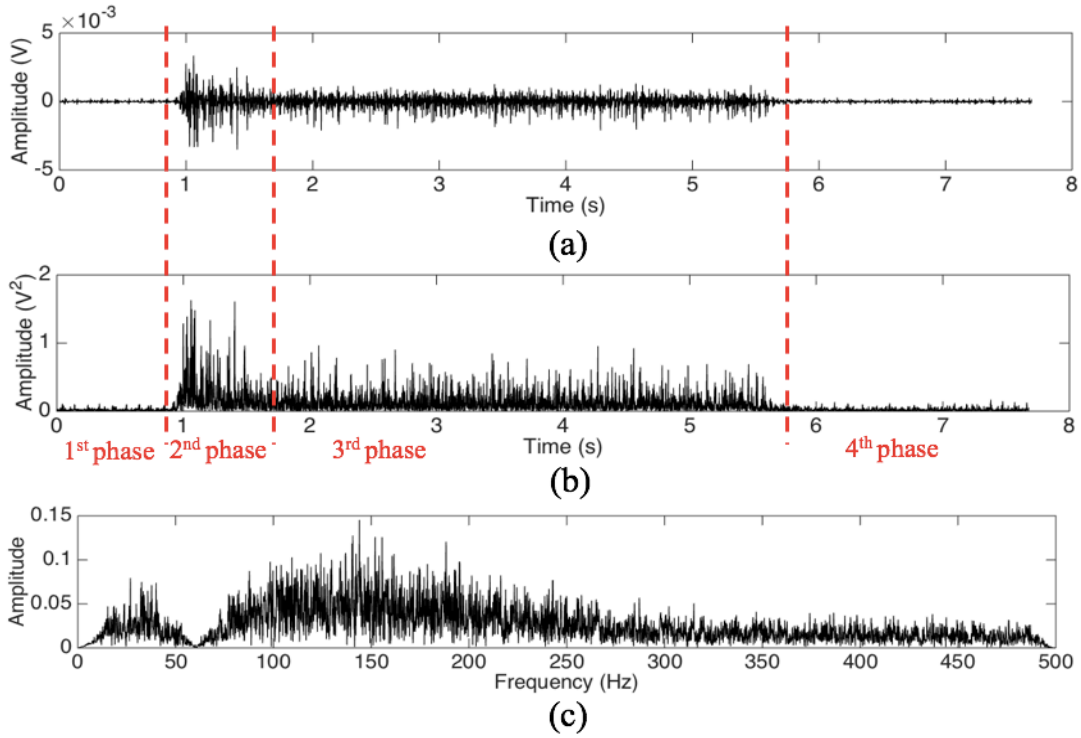

Figure 46 : **(a)** Un signal EMG, **(b)** la forme TKE du même signal, et **(c)** la Transformée de Fourier du signal, pour un mouvement de flexion du pouce sur le canal 1. Le mouvement peut être décomposé en 4 phases : repos, contraction, maintien du mouvement et repos.

Pour chacun des 8 sujets de test, l'efficacité de prédiction a été calculée. Deux méthodes pour ces calculs ont été implémentées : l'efficacité avec la *cross-validation* et avec la fonction *predict* de Matlab.

L'efficacité de *cross-validation* nous donne une moyenne d'efficacité théorique sur l'ensemble de la base de données. L'article [71] explique en détail la méthode de la *cross-validation* qui est très courante pour les calculs d'efficacité dans la littérature. Pour résumer le fonctionnement de cette méthode, supposons que l'on pratique une *k-fold cross-validation* avec k = 10. Toutes les données de la *database* sont séparées en 10 paquets, puis un des paquets est utilisé pour les tests, tandis que les autres servent à entraîner le système et à créer le modèle LDA. Toutes les combinaisons possibles des paquets entre eux sont pratiquées, avec toujours un paquet pour les tests et les autres pour la base de données. On calcule la précision de prédiction pour chaque combinaison possible, puis la moyenne de toutes les précisions est renvoyée. Les fonctions Matlab *crossval* (appliquée à une base de données spécifique) et *kfoldLoss* nous permettent d'obtenir ce résultat.

L'efficacité avec *predict* se calcule en divisant la *database* (10 observations/classe) en une partie de test et une partie d'entraînement. Cette dernière partie nous permet d'entraîner le système et de créer le modèle LDA (*Mdl_train*). Ensuite, on utilise la partie de test pour voir ce que le prédicteur (fonction *predict* de Matlab) nous retourne comme classe pour chaque vecteur. Étant donné que l'on connait les véritables classes associées à ces vecteurs, on peut comparer avec la réponse du prédicteur et ainsi calculer le pourcentage de réussite (efficacité). Cela nous donne une efficacité plus expérimentale. Voici le code utilisé, avec *Xtest* correspondant aux vecteurs des tests et *ytest* aux classes associées à ces vecteurs :

```
%% Efficacité avec predict

    % Efficacité totale
    nb_reussite=0;

    for i=1:2028
    % 2028 pour 2 obs/classe pour les tests et le reste pour l'entraînement

        prediction=predict(Mdl_train,Xtest(i,:));

        if (prediction == ytest(i,:))
            nb_reussite=nb_reussite+1;
        end
    end

    pourcentage_reussite= (nb_reussite/2028)*100;
```

Figure 47 : Code Matlab utilisé pour les calculs d'efficacité avec *predict*.

La Figure 48 montre l'efficacité totale de prédiction, pour chaque sujet avec la *cross-validation* en vert, et avec *predict* en bleu. Cette efficacité va jusqu'à 95,8% au maximum, pour la *cross-validation*, et est en moyenne de 92,7% avec un écart-type de 2,8%. Pour *predict*, l'efficacité maximum est de 96,4%, avec une moyenne de 89,4% et un écart-type de 3,6%. Le sujet 7, ayant des tremblements de main, possède une efficacité moyenne légèrement plus basse que les autres, tout en restant très acceptable pour le contrôle d'une prothèse. L'efficacité moyenne, sur l'ensemble des participants, avec la *cross-validation* est plus élevée que celle avec *predict*, et les valeurs des efficacités sont moins dispersées entre les participants (plus petit écart-type). En effet, l'efficacité avec la *cross-validation*

correspond à une moyenne plus générale du fait de sa méthode de calcul : plusieurs combinaisons de vecteurs d'entraînement/test sont réalisées sur la base de données. Tandis que celle avec *predict* est plus proche de l'efficacité observée en temps réel, car elle met en œuvre une unique combinaison d'entraînement/test sur la base de données. Il est donc intéressant d'avoir accès à ces deux efficacités pour l'analyse des résultats.

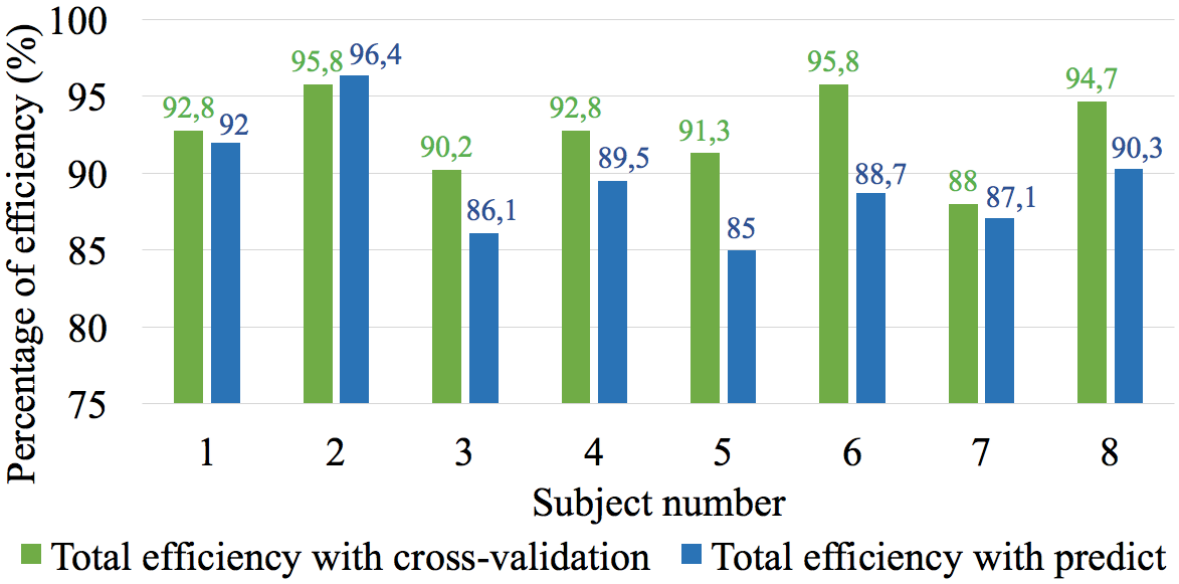

Figure 48 : Pourcentage total d'efficacité de prédiction pour chacun des 8 participants. L'histogramme vert correspond à l'efficacité en utilisant la *cross-validation* et en bleu l'efficacité en utilisant la fonction *predict*.

Le Tableau 8 nous donne l'efficacité de prédiction pour chaque mouvement, avec la fonction *predict*.

Tableau 8 : Efficacités de classification (en %) obtenues pour chacun des participants et des classes (Fig 32) avec la fonction *predict*.

Numéro du sujet / **Mouvement**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Moyenne d'efficacité du mouvement sur tous les participants**	**Écart-type**
Main ouverte	97,4	99,4	99,4	99,4	57,1	100	100	100	**94,1**	**15,0**
Pouce	78,2	99,4	88,5	99,4	97,4	78,2	94,2	100	**91,9**	**9,3**
Index	98,1	100	96,2	99,4	92,3	93,6	84	99,4	**95,4**	**5,4**
Majeur	98,1	100	80,8	91,7	98,7	96,8	94,2	93,6	**94,2**	**6,1**
Annulaire	97,4	97,4	88,5	94,2	78,2	98,7	100	93,6	**93,5**	**7,2**
Auriculaire	91,7	91	67,3	99,4	72,4	94,2	77,6	78,8	**84,1**	**11,5**
Pince simple	89,1	97,4	100	78,8	96,2	97,4	86,5	98,7	**93,0**	**7,5**
Pince complexe	88,5	88,5	87,8	82,7	98,1	65,4	89,7	87,8	**86,1**	**9,4**
Pointer index	85,9	99,4	71,2	78,2	61,5	100	82,7	67,9	**80,9**	**14,1**
3 doigts	98,1	95,5	85,3	81,4	91,7	92,9	77,6	87,2	**88,7**	**7,1**
Main fermée	89,1	86,5	75	71,8	81,4	50,6	73,1	95,5	**77,9**	**13,8**
Prise cylindrique	90,4	100	99,4	98,7	90,4	100	87,8	100	**95,8**	**5,3**

Prise latérale	93,6	98,7	80,1	88,5	89,7	84,6	85,3	71,2	**86,5**	**8,4**
Moyenne d'efficacité du participant sur tous les mouvements	**92,0**	**96,4**	**86,1**	**89,5**	**85,0**	**88,7**	**87,1**	**90,3**	**X**	**X**
Écart-type	**6,0**	**4,7**	**10,9**	**9,9**	**14,0**	**15,3**	**8,4**	**11,2**	**X**	**X**

On peut constater que la plupart des mouvements sont bien reconnus malgré quelques exceptions. La main fermée, l'index pointé et la flexion de l'auriculaire, sont les mouvements qui obtiennent la plus faible efficacité, mais ils ont également un écart-type important. Cela peut signifier que les efficacités moyennes de ces mouvements ont été fortement diminuées par un mauvais score d'un ou deux participants. En règle générale, les écarts-types nous indiquent la dispersion des efficacités autour de leur valeur moyenne. On peut considérer que, malgré quelques exceptions, les efficacités (de l'ensemble des participants) pour chaque mouvement sont peu dispersées (à droite du tableau). Ce qui indique une certaine constance dans la reconnaissance des mouvements entre les participants.

De même, les efficacités (de l'ensemble des mouvements) pour chaque participant sont assez regroupées (écarts-types faibles, en bas du tableau). Cela signifie qu'un même participant parviendra à reconnaître la plupart des mouvements avec une bonne efficacité. Les sujets 5 et 6 ont eu plus de difficultés avec un ou deux mouvements en particulier, d'où les écarts-types plus importants.

Les participants ont des efficacités de prédiction variées comme nous pouvons le constater sur le Tableau 8 et la Figure 48. Cela est dû aux différences de masses musculaires et de tonicités des muscles entre les participants. De plus, chaque personne possède une façon différente de réaliser les mouvements demandés, et parfois, par manque de pratique, le participant a pu rencontrer des difficultés pour produire un mouvement spécifique. Pour chaque sujet, un ou deux mouvements sont souvent moins bien reconnus que les autres même si cela reste acceptable. Il est également important de construire une base de données avec des mouvements d'une intensité et d'une vitesse variées. Car, dans le cas contraire, cela peut être sujet à des erreurs de prédiction en temps réel sur certaines variations d'un mouvement que le système ne peut reconnaître.

En temps réel, quand la *database* est testée, la plupart des sujets parviennent à reproduire les mouvements enregistrés précédemment et le système les identifie correctement. De meilleures efficacités pourraient être obtenues en augmentant le nombre d'observations/mouvement dans la *database*.

Un apprentissage personnel doit également être fait à l'aide du retour d'information sur la prédiction. Nous avons pu constater que lorsque la personne ne parvenait plus à reproduire l'intensité ou la vitesse d'un mouvement enregistré dans la base de données, renvoyer le résultat de la prédiction lui permettait de réajuster son mouvement en temps réel et corriger les éventuelles erreurs. Avec la pratique et la compréhension du fonctionnement du système, il est beaucoup plus facile de reproduire le type de mouvement enregistré.

10 Discussion et travaux futurs

Cette section présente une discussion sur les résultats généraux du projet ainsi qu'une comparaison de notre système avec la littérature et la présentation des travaux futurs. Dans ce projet, nous avons cherché à résoudre la problématique suivante : Comment développer un système de détection de mouvements complexes de la main, qui serait utilisé pour le contrôle d'une prothèse de main myoélectrique ? Ainsi, différents moyens techniques ont été mis en œuvre afin de répondre au mieux à cette problématique.

Tout d'abord, des électrodes sèches de mesure ont été conçues pour enregistrer les signaux EMG à des endroits stratégiques de l'avant-bras. Le type d'électrode, leur nombre et leur placement ont été judicieusement choisis afin d'obtenir les meilleures performances possible. Dans la littérature, d'autres types d'électrodes peuvent être utilisées comme des Ag/AgCl, [33] et [34], qui offrent une bonne qualité de signal, mais qui se détériorent rapidement dans le temps (oxydation des électrodes). De même, les électrodes Delsys DE 2.x sont parfois utilisées [32]. Elles offrent une très bonne qualité de signal, mais restent très coûteuses.

Après cette étape réalisée, une carte d'acquisition et de traitement des signaux EMG a été conçue. Cette carte comprend les éléments essentiels pour faire un enregistrement et un traitement numérique efficace des signaux, comme un microcontrôleur et un composant spécialisé dans l'acquisition de signaux biologiques. Alimenté par une batterie de petite taille, mais de bonne capacité, ce système d'acquisition est autonome (une centaine d'heures) et peut être facilement embarqué sur un utilisateur en comparaison du système de l'article [32].

Utilisant les signaux EMG traités en sortie de la carte d'acquisition, un algorithme de détection de mouvements complexes de la main a été développé sur Raspberry Pi. Cet algorithme permet la classification (avec la méthode LDA) de 13 mouvements fins de la main en temps réel avec une bonne fluidité de prédiction (192ms entre chaque prédiction). Là encore, l'avantage majeur de l'utilisation de la Raspberry Pi est qu'il est possible d'embarquer le système en totalité sur un utilisateur, à l'aide d'une batterie d'une grande autonomie (plus de 100 heures également).

Afin d'aller plus loin dans l'analyse du système développé, des tests ont été réalisés pour déterminer les performances dans la détection des mouvements de la main. L'algorithme de détection ayant également été développé sous Matlab, nous avons mis en place un protocole expérimental. Le système a été testé sur 8 participants, d'âges et de conditions physiques variées, afin de déterminer les efficacités de prédiction des différents mouvements de la main. Il leur a été demandé d'enregistrer une base de données contenant plusieurs enregistrements des mouvements de la main à classifier, afin de pouvoir entraîner le système à chacun des patients. L'interface Matlab, prévue à cet effet, a permis de faciliter la construction de la base de données. Après cela, la fluidité et la validité de la prédiction en temps réel ont pu être testées à l'aide d'une autre interface Matlab, développée pour avoir une rétroaction des mouvements prédits. Pour finir, les efficacités de prédiction ont été calculées pour l'ensemble des mouvements de la main à détecter. Nous avons pu atteindre une efficacité allant jusqu'à 95,8% au maximum, et en moyenne de 92,7% avec un écart-type

de 2,8%. Nous pouvons comparer l'efficacité de notre système avec les efficacités obtenues dans les articles [33], [34] et [72]. Dans l'article [33], une efficacité de 92,1% a été obtenue pour la détection de six mouvements de la main sur 20 participants sains. Cinq canaux EMG ont été utilisés et la classification a été réalisée à l'aide de la méthode LDA et des coefficients AR d'ordre 4. Tandis qu'une efficacité moyenne de 90% a été trouvée dans [34], sur 10 participants sains avec cette même méthode LDA et les coefficients AR d'ordre 4. L'expérience ayant été faite pour la reconnaissance de 8 mouvements de la main avec 4 canaux EMG. Kim et al. [72] ont obtenu une efficacité de 81,1% en classifiant 4 mouvements de la main avec la méthode LDA et trois paramètres discriminants temporels (*Integrated Absolute Value*, *Difference Absolute Mean* et *Difference Absolute Standard Deviation*). Deux canaux EMG sont utilisés et 30 participants en santé ont participé à cette étude. Nous pouvons donc considérer que l'efficacité moyenne de 92,7%, obtenue avec notre système, est très acceptable.

Après analyse des efficacités moyennes et des écarts-types de chacun des mouvements pour les participants, nous avons constaté que la plupart des mouvements étaient bien reconnus et qu'on observait une certaine constance dans la reconnaissance des mouvements entre les participants. De plus, un même participant parviendra à reconnaître la plupart des mouvements avec une bonne efficacité. L'expérience a été globalement une réussite, car le système est parvenu à identifier correctement les mouvements en temps réel pour la plupart des sujets. Nous avons souhaité que le système de détection soit personnel à chaque utilisateur (base de données différente pour chaque sujet), car chaque personne a une façon unique de produire les signaux EMG, ainsi le système sera mieux adapté à chaque usager et la détection de mouvement plus efficace.

Pour aller plus loin, un prototype de prothèse mécatronique a été développé. Une main mécanique imprimée en 3D a été motorisée à l'aide de trois moteurs/réducteurs/encodeurs de la gamme Maxon. Un moteur contrôle le pouce, un autre l'index et le majeur, et le dernier l'annulaire et l'auriculaire. De plus, l'ensemble des éléments, incluant les contrôleurs des moteurs, a pu être embarqué dans un boîtier rattaché à la main. L'objectif étant que le contrôle de la position des moteurs soit effectué en fonction du mouvement prédit par l'algorithme sur Raspberry Pi. Le coût global de ce prototype a pu être minimisé par l'utilisation de matières peu coûteuses (impression avec une imprimante 3D de la plupart des pièces). Le prix final est, de ce fait, bien inférieur au prix des prothèses commerciales courantes (voir les Tableaux 1 et 7).

Pour conclure la discussion, le principal objectif de ce projet a été atteint par la réalisation d'un nouveau système de détection de mouvements complexes de la main. L'utilisateur pouvant avoir accès à un panel de mouvements utiles pour contrôler sa prothèse mécatronique. De plus, les accents ont été portés sur la nécessité d'avoir une prédiction de mouvement fluide, rapide et très intuitive pour l'utilisateur. Ce prototype de système de contrôle embarqué utilise des éléments peu coûteux afin de minimiser le coût global de la prothèse, et prédit les mouvements de la main avec une très bonne efficacité.

Ainsi, nous avons répondu à la problématique du projet même s'il restera toujours des améliorations futures à développer. La suite de cette section a pour objectif de les mettre en évidence.

- Étudier la façon de commander les moteurs de façon optimale : à l'aide du retour de prédiction de la Raspberry Pi, il faudra imposer une tension analogique pour contrôler chaque moteur. Analyser la position voulue des moteurs, pour effectuer le mouvement adéquat de la main, est nécessaire (contrôle en position, vitesse, asservissement…).

- Créer un ou des contrôleurs pour les moteurs : en effet, nous avons utilisé des contrôleurs commerciaux dans ce projet. Il pourrait être envisagé, pour la suite, de concevoir un unique contrôleur, pour les 3 moteurs, contenant l'ensemble des éléments utiles à ce projet. Cela pourrait notamment réduire le volume et le coût de la prothèse.

- Augmenter la puissance de calcul de la Raspberry Pi : nous pourrions changer de processeur et créer une carte spéciale contenant uniquement les éléments nécessaires à ce projet. En effet, la Pi 3 possède un bon processeur, mais beaucoup trop d'éléments inutiles (USB, connecteur RJ45…), ce qui augmente inutilement le volume de la carte.

- Considérant les avancées en matière de puissance de calcul grandissante des systèmes embarqués, nous pourrions envisager d'utiliser une méthode de classification plus complexe plutôt que la méthode LDA. Car même si elle reste une méthode sûre, rapide et robuste, de nouvelles techniques voient le jour telle que le développement de la classification par réseaux de neurones.

- Utilisation du module RF sur la carte d'acquisition : Nous pourrions établir une communication sans fil entre la carte d'acquisition et la Raspberry Pi Zero W (connectée aux contrôleurs des moteurs).

- Isolation de la carte d'acquisition : nous avons pu observer que cette carte était parfois perturbée par les champs électromagnétiques à proximité. Par exemple : des téléphones portables, d'autres appareils branchés à l'ordinateur de mesure (câbles Ethernet …), etc.

- Isolation des câbles reliant les électrodes à la carte d'acquisition : nous utilisons actuellement de simples câbles femelles/femelles (Jumpers). Étant donné le nombre de câbles utilisés, ils ont tendance à s'entrechoquer, et se déplacer. Nous pourrions les regrouper dans un seul câble (avec une bonne isolation) de plus gros diamètre contenant plusieurs lignes à l'intérieur (Octopaire).

- Maintien de la position des électrodes : il faudrait coudre les électrodes sur un support qu'on puisse enfiler (attèle, manche en tissu…) afin d'avoir une configuration fixe des électrodes. Cela permettrait également de gagner beaucoup de temps au niveau du collage des électrodes. Mais attention, la configuration des électrodes peut être

différente selon les sujets et la position des muscles (différentes masses musculaires, etc.). Par exemple, les prothèses commerciales ajustent la position des électrodes aux différents patients en faisant un moulage différent de l'emboîture.

- Achat d'une batterie pour les moteurs et la Raspberry Pi : pour le moment, une batterie alimente la Pi et des alimentations stabilisées sont utilisées pour les moteurs. Mais il faudrait étudier la consommation nécessaire pour alimenter les moteurs et la Raspberry Pi afin d'ajouter une (ou plusieurs) batterie de taille adaptée au système embarqué.

- Achat d'un écran tactile miniature pour le retour de prédiction et la commande tactile de l'algorithme. De petits écrans spécifiques à la Raspberry Pi peuvent être trouvés dans le commerce.

- Portage du code de la construction de la base de données sur la Pi : nous pourrions alors envisager de nous passer de l'interface graphique sous Matlab.

- Penser à la façon d'embarquer l'ensemble des éléments sur un usager. La Raspberry Pi devra être connectée aux contrôleurs des moteurs présents dans le boîtier de la prothèse. Nous pourrions également trouver la batterie et la carte d'acquisition proches du boîtier.

- Pour des personnes amputées, certains muscles ont pu être endommagés ou se trouvent à des endroits différents de l'avant-bras. Il faudra donc placer judicieusement les électrodes en fonction des différents cas d'amputations et pas nécessairement reproduire le même scénario que dans cette étude. De même, l'usager devra être capable de contracter des muscles spécifiques, donc un entraînement préalable peut être nécessaire. On peut notamment décider de choisir d'autres muscles où placer les électrodes si la contraction d'un muscle spécifique est trop compliquée.

11 Conclusion :

Dans ce mémoire, une approche pour la détection de mouvements complexes de la main a été mise en place. Des électrodes sèches ont été conçues et judicieusement placées sur l'avant-bras afin de recueillir les signaux EMG utiles à notre système. Une carte d'acquisition multicanale, spécialement adaptée à notre application, nous a permis d'effectuer un traitement efficace de ces signaux. C'est grâce à la qualité de ceux-ci qu'une classification des mouvements de la main a pu être établie. Pour cette classification, la méthode LDA a été mise en place, d'abord à l'aide du logiciel Matlab sous lequel nous avons développé des interfaces graphiques abordables pour les futurs utilisateurs. Enfin, l'algorithme de prédiction des mouvements a été porté sur une Raspberry Pi Zero afin de rendre notre système embarqué. En parallèle de cela, nous avons motorisé une main 3D dans le but de pouvoir l'utiliser avec le système de détection des mouvements de la main.

La méthode établie nous a permis de distinguer la plupart des mouvements de la main avec une très bonne efficacité de prédiction. Cette efficacité pouvant aller jusqu'à 95,8% et en moyenne de 92,7% sur les 8 personnes ayant participé aux tests de notre système. Celui-ci permet notamment une actualisation de la prédiction toutes les 192ms avec une détection de 13 mouvements différents. De plus, le système a été développé dans l'optique de pouvoir être embarqué sur un utilisateur tout en ayant un prix abordable.

Il reste encore des améliorations à apporter à ce projet complexe et celui-ci reste ouvert à des développements futurs. Rendre les prothèses myoélectriques actuelles plus intuitives a été la motivation première de ce projet et le prototype développé semble être une bonne solution pour contrôler les prothèses myoélectriques de demain.

Bibliographie :

[1] K. Ziegler-Graham, E. J. MacKenzie, P. L. Ephraim, T. G. Travison et R. Brookmeyer, «Estimating the Prevalence of Limb Loss in the United States: 2005 to 2050.,» *Archives of Physical Medicine and Rehabilitation,* vol. 89, n° 3, p. 422-9, 2008.

[2] A. Coalition. [En ligne]. Available: https://www.amputee-coalition.org/limb-loss-resource-center/resources-filtered/resources-by-topic/limb-loss-statistics/limb-loss-statistics/#1. [Accès le 05 11 2018].

[3] I. S. &. H. N. (ISHN), «Statistics on hand and arm loss» 04 02 2014. [En ligne]. Available: https://www.ishn.com/articles/97844-statistics-on-hand-and-arm-loss. [Accès le 05 11 2018].

[4] Ottobock, «Ottobock North America Consumer Home | Ottobock US» 2017. [En ligne]. Available: https://www.ottobockus.com. [Accès le 02 11 2018].

[5] T. B. b. Össur, «Home | Touch Bionics» 2018. [En ligne]. Available: http://www.touchbionics.com. [Accès le 02 11 2018].

[6] Bebionic Ottobock, «Sujet : Bebionic prosthesis» [En ligne]. Available: http://bebionic.com. [Accès le 15 Janvier 2018].

[7] P. Geethanjali, «Myoelectric control of prosthetic hands: state-of-the-art review,» *Medical Devices (Auckland, N. Z.),* vol. 9, pp. 247-255, 2016.

[8] B. Peerdeman et al., «Myoelectric forearm prostheses: State of the art from a user-centered perspective,» *Journal of Rehabilitation Research and Development (JRRD),* vol. 48, n° 6, pp. 719-738, 2011.

[9] R. Crepin, C. L. Fall, Q. Mascret, C. Gosselin, A. Campeau-Lecours et B. Gosselin, «Real-Time Hand Motion Recognition Using sEMG Patterns Classification,» *40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC),* pp. 239-242, 2018.

[10] C. Orthopédie, «Sujet : Les différentes prothèses myoélectriques» [En ligne]. Available: https://www.chabloz-ortho.com/fr/prothese/prothese-membre-superieur/prothese-myoelectrique/. [Accès le 15 Novembre 2018].

[11] I-Limb Touch Bionics by Össur, «Sujet : I-Limb prosthesis» [En ligne]. Available: https://www.touchbionics.com/products/how-i-limb-works. [Accès le 15 Janvier 2018].

[12] W. D. Memberg, T. G. Stage et R. F. Kirsch, «A fully implanted intramuscular bipolar myoelectric signal recording electrode,» *Neuromodulation: Technology at the Neural Interface,* vol. 17, n° 8, pp. 794-799, 2014.

[13] G. Rognini et al., «Multisensory bionic limb to achieve prosthesis embodiment and reduce distorted phantom limb perceptions,» *Journal of Neurology Neurosurgery and Psychiatry,* 2018.

[14] T. W. Williams III, «Progress on stabilizing and controlling powered upper-limb prostheses,» *Journal of Rehabilitation Research and Development,* vol. 48, n° 6, 2011.

[15] A. Ninu et al., «Closed-Loop Control of Grasping With a Myoelectric Hand Prosthesis: Which Are the Relevant Feedback Variables for Force Control?,» *IEEE Transactions on Neural Systems and Rehabilitation Engineering,* vol. 22, n° 5, pp. 1041-1052, 2014.

[16] E. D'Anna et S. Micera, «Designing sensory feedback approaches for restoring touch and position feedback in upper limb amputees,» *Infoscience EPFL scientific publications, Thesis,* 2018.

[17] I. Vujaklija, «Novel Control Strategies for Upper Limb Prosthetics,» *Converging Clinical and Engineering Research on Neurorehabilitation III, ICNR,* pp. 171-174, 2018.

[18] M. Aboseria, F. Clemente, L. F. Engels et C. Ciprian, «Discrete Vibro-Tactile Feedback Prevents Object Slippage in Hand Prostheses More Intuitively Than Other Modalities,» *IEEE Transactions on Neural Systems and Rehabilitation Engineering,* vol. 26, n° 8, pp. 1577 - 1584, 2018.

[19] A. W. Shehata et al., «Improving internal model strength and performance of prosthetic hands using augmented feedback,» *Journal of NeuroEngineering and Rehabilitation,* vol. 15, n° 70, 2018.

[20] P. Svensson, U. Wijk, A. Björkman et C. Antfolk, «A review of invasive and non-invasive sensory feedback in upper limb prostheses,» *Expert Review of Medical Devices,* vol. 14, n° 6, pp. 439-447, 2017.

[21] M. Schiefer, D. Tan, S. M. Sidek et D. J. Tyler, «Sensory feedback by peripheral nerve stimulation improves task performance in individuals with upper limb loss using a myoelectric prosthesis,» *Journal of Neural Engineering,* vol. 13, n° 1, 2015.

[22] C. Hartmann, S. Došen, S. Amsuess et D. Farina, «Closed-Loop Control of Myoelectric Prostheses With Electrotactile Feedback: Influence of Stimulation Artifact and Blanking,» *IEEE Transactions on Neural Systems and Rehabilitation Engineering,* vol. 23, n° 5, pp. 807-816, 2015.

[23] F. Clemente, M. D'Alonzo, M. Controzzi, B. B. Edin et C. Cipriani, «Non-Invasive, Temporally Discrete Feedback of Object Contact and Release Improves Grasp Control of Closed-Loop Myoelectric Transradial Prostheses,» *IEEE Transactions on Neural Systems and Rehabilitation Engineering,* vol. 24, n° 12, pp. 1314-1322, 2015.

[24] M. Tommerdahl et al., «Human vibrotactile frequency discriminative capacity after adaptation to 25 Hz or 200 Hz stimulation,» *Brain Research,* vol. 1057, n° 1-2, pp. 1-9, 2005.

[25] R. Barone et al., «Multilevel control of an anthropomorphic prosthetic hand for grasp and slip prevention,» *Advances in Mechanical Engineering,* vol. 8, n° 9, 2016.

[26] S. Schulz et al., «Design and Preliminary Experience with Fluidhand MK III,» *Proceedings of the 2008 MyoElectric Controls/Powered Prosthetics Symposium,* 2008.

[27] B. Hudgins, P. Parker et R. N. Scott, «A new strategy for multifunction myoelectric control,» *IEEE Transactions on Biomedical Engineering,* vol. 40, n° 1, pp. 82-94, 1993.

[28] L. H. Smith, T. A. Kuiken et L. J. Hargrove, «Real-time simultaneous and proportional myoelectric control using intramuscular EMG,» *Journal of Neural Engineering,* vol. 11, n° 6, 2014.

[29] N. Jiang, J. L. Vest-Nielsen, S. Muceli et D. Farina, «EMG-based simultaneous and proportional estimation of wrist/hand kinematics in uni-lateral trans-radial amputees,» *Journal of NeuroEngineering and Rehabilitation,* vol. 9, n° 42, 2012.

[30] M. W. Johnson et P. H. Peckham, «Evaluation of Shoulder Movement as a Command Control Source,» *IEEE Transactions on Biomedical Engineering,* vol. 37, n° 9, pp. 876-885, 1990.

[31] R. Branemark, P. I. Branemark, B. Rydevik et R. R. Myers, «Osseointegration in skeletal reconstruction and rehabilitation: A review,» *Journal of Rehabilitation Research and Development,* vol. 38, n° 2, pp. 175-181, 2001.

[32] R. N. Khushaba et S. Kodagoda, «Electromyogram (EMG) Feature Reduction Using Mutual Components Analysis for Multifunction Prosthetic Fingers Control,» *Control Automation Robotics & Vision (ICARCV),* 2012.

[33] A. Phinyomark, P. Phukpattaranont et C. Limsakul, «Feature reduction and selection for EMG signal classification,» *Expert Systems with Applications,* vol. 39, n° 8, pp. 7420-7431, 2012.

[34] F. Al Omari, J. Hui, C. Mei et G. Liu, «Pattern Recognition of Eight Hand Motions Using Feature Extraction of Forearm EMG Signal,» *Proceedings of the National Academy of Sciences,* vol. 84, n° 3, pp. 473-480, 2014.

[35] S. Solnik, P. Rider, K. Steinweg, P. DeVita et T. Hortobagyi, «Teager-Kaiser energy operator signal conditioning improves EMG onset detection,» *European Journal of Applied Physiology,* vol. 110, n° 3, pp. 489-498, 2010.

[36] A. Tharwat, A. Ibrahim, T. Gaber et A. E. Hassanien, «Linear discriminant analysis: A detailed tutorial,» *AI Communications,* vol. 30, n° 2, pp. 169-190, 2017.

[37] N. Wang, Y. Chen et X. Zhang, «The recognition of multi-finger prehensile postures using LDA,» *Biomedical Signal Processing and Control,* vol. 8, pp. 706-712, 2013.

[38] S. Amsuess, P. Goebel, B. Graimann et D. Farina, «A Multi-Class Proportional Myocontrol Algorithm for Upper Limb Prosthesis Control: Validation in Real-Life Scenarios on Amputees,» *IEEE Transactions Neural Systems and Rehabilitation Engineering,* vol. 23, pp. 827-836, 2014.

[39] K. Xing, P. Yang, J. Huang, Y. Wang et Q. Zhu, «A real-time EMG pattern recognition method for virtual myoelectric hand control,» *Neurocomputing,* vol. 136, pp. 345-355, 2014.

[40] A. J. Young, L. H. Smith, E. J. Rouse et L. J. Hargrove, «Classification of Simultaneous Movements Using Surface EMG Pattern Recognition,» *IEEE Transactions on Biomedical Engineering,* vol. 60, n° 5, pp. 1250-1258, 2013.

[41] I. Mesa, A. Rubio, I. Tubia, J. De No et J. Diaz, «Channel and feature selection for a surface electromyographic pattern recognition task,» *Expert Systems with Applications,* vol. 41, pp. 5190-5200, 2014.

[42] J. L. Pons et al., «Virtual reality training and EMG control of the MANUS hand prosthesis,» *Robotica,* vol. 23, n° 3, pp. 311-317, 2005.

[43] D. P. Yang et al., «An anthropomorphic robot hand developed based on underactuated mechanism and controlled by EMG signals,» *Journal of Bionic Engineering,* vol. 6, n° 3, pp. 255-263, 2009.

[44] The Academy AAOP (American Academy of Orthotists & Prosthetists), «Sujet : FluidHand III» [En ligne]. Available: http://www.oandp.org/jpo/library/2009_02_091.asp. [Accès le 1 Novembre 2016].

[45] C. M. Light, P. H. Chappell, B. Hudgins et K. Engelhart, «Intelligent multifunction myoelectric control of hand prosthesis,» *Journal of Medical Engineering & Technology,* vol. 26, n° 4, pp. 139-146, 2002.

[46] G. Matrone, C. Cipriani, E. L. Secco, M. C. Carrozza et G. Magenes, «Bio-inspired controller for a dexterous prosthetic hand based on principal components analysis,» *Conference Proceedings, IEEE Engineering in Medicine and Biology Society,* pp. 5022-5025, 2009.

[47] C. Cipriani, F. Zaccone, S. Micera et M. C. Carrozza, «On the shared control of an EMG-controlled prosthetic hand: Analysis of user-prosthesis interaction,» *IEEE Transactions on Robotics,* vol. 24, n° 1, pp. 170-184, 2008.

[48] Community Research and Development Information Service (CORDIS), «Sujet : CyberHand» [En ligne]. Available: http://cordis.europa.eu/project/rcn/63258_en.html. [Accès le 10 Mai 2017].

[49] O. Christ et al., «Prosthesis-user-in-the-loop: User-centered design parameters and visual simulation,» *Annual International Conference of the IEEE Engineering in Medicine and Biology Society,* pp. 1929-1932, 2012.

[50] E. Mainardi et A. Davalli, «Controlling a prosthetic arm with a throat microphone,» *29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society,* pp. 3035-9, 2007.

[51] C. C. Postelnicu, D. Talaba et M. I. Toma, «Controlling a Robotic Arm by Brainwaves and Eye Movement,» *Technological Innovation for Sustainability, DoCEIS 2011,* pp. 157-164, 2011.

[52] J. Gonzalez et W. Yu, «Multichannel audio aided dynamical perception for prosthetic hand biofeedback,» *IEEE International Conference on Rehabilitation Robotics,* 2009.

[53] J. E. Cheesborough, L. H. Smith, T. A. Kuiken et G. A. Dumanian, «Targeted Muscle Reinnervation and Advanced Prosthetic Arms,» *Seminars in Plastic Surgery,* vol. 29, n° 1, pp. 62-72, 2015.

[54] Le Huffington Post, «Une main artificielle commandée par la pensée, dernière invention des chercheurs de l'Université de Pittsburgh» [En ligne]. Available: http://www.huffingtonpost.fr/2012/12/17/bionique-une-main-artificielle_n_2314072.html. [Accès le 1 Novembre 2016].

[55] A. M. Barber et D. L. Juan, «Sensor for acquiring physiological signals,» *US Patent (Patent # 9,629,584),* 2017.

[56] Y. Zhu, A. Noda, Y. Makino et H. Shinoda, «Myoelectric pattern measurement on a forearm via conductive fabric,» *Proceedings of the SICE (Society of Instrument and Control Engineers) Annual Conference 2017, Japan,* pp. 966-969, 2017.

[57] C. L. Fall, G. Gagnon-Turcotte, J. F. Dube, J. S. Gagne, Y. Delisle, A. Campeau-Lecours, C. Gosselin et B. Gosselin, «A Wireless sEMG-Based Body-Machine Interface for Assistive Technology Devices,» *IEEE Journal of Biomedical and Health Informatics,* vol. 21, n° 4, pp. 967-977, 2017.

[58] C. L. Fall, P. Turgeon, V. Maheu, A. Campeau-Lecours, M. Boukadoum, S. Roy, D. Massicotte, C. Gosselin et B. Gosselin, «Intuitive wireless control of a robotic arm for people living with an upper body disability,» *The 37th Annual International*

Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), pp. 4399 - 4402, 2015.

[59] C. L. Fall, F. Quevillon, A. Campeau-Lecours, S. Latour, M. Blouin, C. Gosselin et B. Gosselin, «A Multimodal Adaptive Wireless Control Interface for People with Upper-Body Disabilities,» *IEEE International Symposium on Circuits and Systems (ISCAS),* pp. 1-4, 2017.

[60] C. L. Fall et al., «A multimodal adaptive wireless control interface for people with upper-body disabilities,» *IEEE Transactions on Biomedical Circuits and Systems,* vol. 12, n° 3, p. 564–575, June 2018.

[61] B. Gosselin et M. Sawan, «An Ultra Low-Power CMOS Automatic Action Potential Detector,» *IEEE Transactions on Neural Systems and Rehabilitation Engineering,* vol. 17, pp. 346-353, 2009.

[62] U. Côté-Allard, C. L. Fall, A. Campeau-Lecours, C. Gosselin, F. Laviolette et B. Gosselin, «Transfer learning for sEMG hand gestures recognition using convolutional neural networks,» *IEEE International Conference on Systems, Man, and Cybernetics (SMC),* pp. 1663-1668, 2017.

[63] D. Tkach, H. Huang et T. A. Kuiken, «Study of stability of time-domain features for electromyographic pattern recognition,» *Journal of NeuroEngineering and Rehabilitation,* vol. 7, n° 21, 2010.

[64] M. Zardoshti-Kermani, B. C. Wheeler, K. Badie et R. M. Hashemi, «EMG feature evaluation for movement control of upper extremity prostheses,» *IEEE Transactions on Rehabilitation Engineering,* vol. 3, n° 4, pp. 324-333, 1995.

[65] J. R. Mathiesen et al., «Prediction of grasping force based on features of surface and intramuscular EMG,» *7th semester conference paper,* vol. 1, n° 9, 2010.

[66] G. Eshel, «The Yule Walker Equations for the AR Coefficients» [En ligne]. Available: http://www-stat.wharton.upenn.edu/~steele/Courses/956/ResourceDetails/YWSourceFiles/YW-Eshel.pdf. [Accès le 03 11 2018].

[67] A. Phinyomark et al., «EMG feature evaluation for improving myoelectric pattern recognition robustness,» *Expert Systems with Applications,* vol. 40, n° 12, pp. 4832-4840, 2013.

[68] B. Şen, M. Peker, A. Çavuşoğlu et F. V. Çelebi, «A Comparative Study on Classification of Sleep Stage Based on EEG Signals Using Feature Selection and Classification Algorithms,» *Journal of Medical Systems,* vol. 38, n° 18, 2014.

[69] S. Solnik, P. DeVita, P. Rider, B. Long et T. Hortobágyi, «Teager–Kaiser Operator improves the accuracy of EMG onset detection independent of signal-to-noise ratio,» *Acta of Bioengineering and Biomechanics,* vol. 10, n° 2, pp. 65-68, 2008.

[70] M. Baril, T. Laliberté, C. Gosselin et F. Routhier, «On the design of a mechanically programmable underactuated anthropomorphic prosthetic gripper,» *Journal of Mechanical Design,* vol. 135, n° 12, pp. 121008/1-121008/9, 2013.

[71] T.-T. Wong, «Performance evaluation of classification algorithms by k-fold and leave-one-out cross validation,» *Pattern Recognition,* vol. 48, n° 9, pp. 2839-2846, 2015.

[72] K. S. Kim, H. H. Choi, C. S. Moon et C. W. Mun, «Comparison of k-nearest neighbor, quadratic discriminant and linear discriminant analysis in classification of electromyogram signals based on the wrist-motion directions,» *Current Applied Physics,* vol. 11, pp. 740-745, 2011.

[73] N. Wang, Y. Chen et X. Zhang, «Realtime recognition of multi-finger prehensile gestures,» *Biomedical Signal Processing and Control,* vol. 13, pp. 262-269, 2014.

[74] P. Herberts, C. Almström, R. Kadefors et P. D. Lawrence, «Hand prosthesis control via myoelectric patterns,» *Acta Orthopaedica Scandinavica,* vol. 44, n° 4, pp. 389-409, 1973.

Annexe A : Code Matlab des interfaces graphiques (Partie 6)

Dans cette annexe, les codes développés pour l'interface graphique Matlab de la prédiction en temps réel (Partie 6.6) ainsi que l'interface pour la construction de la base de données (Partie 6.5), seront présentés. L'ensemble du fonctionnement des boutons de l'interface ne sera pas détaillé, mais l'essentiel sera mis en valeur.

Pour cette annexe, les codes refléteront les calculs et les traitements pour un seul canal afin de faciliter la compréhension générale. Les conditions utilisées dans les boucles et les étapes importantes liées à la classification/prédiction des mouvements tiendront compte de l'ensemble des canaux afin de se placer en condition réelle. C1 correspondra au canal 1, C2 au canal 2 … C7 au canal 7.

Les routines Matlab des deux interfaces sont répétées de façon périodique après le lancement du *timer* Matlab. Seule la routine Matlab de l'interface de prédiction en temps réel sera détaillée ici. En effet, la routine de l'interface pour la construction de la base de données correspond en tout point à la partie fléchée (flèches rouges) de la première routine.

De même, les initialisations des deux interfaces étant quasiment identiques, seules les initialisations pour l'interface de prédiction en temps réel seront détaillées ici (flèches rouges pour les initialisations de l'interface pour la construction de la base de données).

Routine Matlab pour l'interface de prédiction en temps réel :

```
function routine(hObject, eventdata, handles)

global img_pouce img_index img_majeur img_annulaire img_auriculaire
 img_flexion_3doigts
global img_pointer_index img_pince_simple img_pince_complexe
 img_prise_cylindrique img_prise_laterale
global img_main_fermee img_main_ouverte

global xnm11

global Y1_C1 Y2_C1 Y3_C1 Y4_C1 Z1_C1 Z2_C1 Z3_C1 Z4_C1
global W1_C1 W2_C1 X1_C1 X2_C1

global NUM_BP DEN_BP
global NUM_BS DEN_BS

global indice

global data_C1_load
global TKE_C1

global board
global flag

global XB_bytes
global NB_bytes_to_read

global echant_canaux paquet nb_canaux
global taille_paquet

global Mdl
global doigt_gagnant
global seuil seuil2 seuil3 seuil4 seuil5 seuil6 seuil7


%% Acquisition
    while (flag==0)
% Quand on réinit ou qu'on appuie sur démarrer

        if (board.BytesAvailable <= XB_bytes)
            return
        end

% On lit un buffer de taille XB_bytes initialisé à NB_bytes_to_read
        data_in = fread(board, XB_bytes);
        i=1;

        while (1)
% Effectue la boucle tant qu'on ne rencontre pas de 1 suivi de 254
            if (data_in(i)==1 && data_in(i+1)==254 &&
             data_in(i+taille_paquet*1)==1 && data_in(i+1+taille_paquet*1)==254 &&
             data_in(i+taille_paquet*2)==1 && data_in(i+1+taille_paquet*2)==254 &&
             data_in(i+taille_paquet*3)==1 && data_in(i+1+taille_paquet*3)==254 &&
             data_in(i+taille_paquet*4)==1)
                break;
            end
            i=i+1;
```

```
        end

% On a trouvé le 1 (en position i)
        Nb_bit_manquant=i;
% Donc il manque i bit pour avoir 6 paquets complets (+ un début de paquet
 inutile)
        if (data_in(XB_bytes)==1 && data_in(XB_bytes-taille_paquet+1)==254 &&
         data_in(XB_bytes-taille_paquet)==1 && data_in(XB_bytes-
         (taille_paquet*2)+1)==254 && data_in(XB_bytes-(taille_paquet*2))==1 &&
         data_in(XB_bytes-(taille_paquet*3)+1)==254 && data_in(XB_bytes-
         (taille_paquet*3))==1 && data_in(XB_bytes-(taille_paquet*4)+1)==254 &&
         data_in(XB_bytes-(taille_paquet*4))==1 && data_in(XB_bytes-
         (taille_paquet*5)+1)==254 && data_in(XB_bytes-(taille_paquet*5))==1 &&
         data_in(XB_bytes-(taille_paquet*6)+1)==254)
            flag=1;
% On peut passer à la suite des traitements
        else
% Pour avoir 6 paquets complets, on rajoute les bits manquants et on refait la
 boucle une fois
            XB_bytes=NB_bytes_to_read+Nb_bit_manquant;
        end
    end

    if (board.BytesAvailable <= NB_bytes_to_read)
        return
    end

% On lit un nouveau buffer de taille initiale (NB_bytes_to_read) mais cette fois
 avec 254 au début
% car le buffer incomplet a été effacé dans la boucle d'avant afin de tomber sur
 un nombre pile de 6 paquets
    data_in = fread(board, NB_bytes_to_read);
% On ne prend pas le 254 pour data_in_corrige
    data_in_corrige=data_in(2:end);


%% Dans cette partie
% On réorganise les canaux avec 64 moitiés d'échantillons/canal X6
% paquets = 384 moitiés d'échantillons/canal en tout
    if (data_in_corrige((echant_canaux*nb_canaux*1)+1) == 1 &&
     data_in_corrige((echant_canaux*nb_canaux*1)+2) == 254 &&
     data_in_corrige((echant_canaux*nb_canaux*2)+2+1) == 1 &&
     data_in_corrige((echant_canaux*nb_canaux*2)+2+2) == 254 &&
     data_in_corrige((echant_canaux*nb_canaux*3)+4+1) == 1 &&
     data_in_corrige((echant_canaux*nb_canaux*3)+4+2) == 254 &&
     data_in_corrige((echant_canaux*nb_canaux*4)+6+1) == 1 &&
     data_in_corrige((echant_canaux*nb_canaux*4)+6+2) == 254 &&
     data_in_corrige((echant_canaux*nb_canaux*5)+8+1) == 1 &&
     data_in_corrige((echant_canaux*nb_canaux*5)+8+2) == 254 &&
     data_in_corrige((echant_canaux*nb_canaux*6)+10+1) == 1)

        for c=1:(echant_canaux*paquet)

            if (c<(echant_canaux+1))
% Paquet 1 de 1 à 64
                data_in_canal_1_bis(c)=data_in_corrige(c);
```

```
            elseif (c>echant_canaux && c<(2*echant_canaux+1))
% Paquet 2 de 65 à 128
                data_in_canal_1_bis(c)=data_in_corrige(c+(echant_canaux*6)+2);

            elseif (c>2*echant_canaux && c<(3*echant_canaux+1))
% Paquet 3 de 129 à 192
                data_in_canal_1_bis(c)=data_in_corrige(c+(echant_canaux*12)+4);

            elseif (c>3*echant_canaux && c<(4*echant_canaux+1))
% Paquet 4 de 193 à 256
                data_in_canal_1_bis(c)=data_in_corrige(c+(echant_canaux*18)+6);

            elseif (c>4*echant_canaux && c<(5*echant_canaux+1))
% Paquet 5 de 257 à 320
                data_in_canal_1_bis(c)=data_in_corrige(c+(echant_canaux*24)+8);

            elseif (c>5*echant_canaux && c<(6*echant_canaux+1))
% Paquet 6 de 321 à 384
                data_in_canal_1_bis(c)=data_in_corrige(c+(echant_canaux*30)+10);
            end
        end
% Si pas de 1 ou de 254 alors l'enregistrement est faux et on recommence
    else return;
    end

%% Regroupement
% Les données sont dans le bon ordre et triées par canaux, maintenant il faut
 regrouper les 2
% bytes par échantillon en utilisant le complément à 2. Ainsi les
% data_in_canal_x seront sur 16 bits

    for b=1:(echant_canaux*paquet)/2
        data_in_canal_1(b) = Complement2(bitsll(data_in_canal_1_bis((b-1)*2+1),8)
         + data_in_canal_1_bis((b-1)*2+2));
    end

%% Initialisation pour boucle
    data_in_canal_1(((echant_canaux*paquet)/
     2)+1)=data_in_canal_1((echant_canaux*paquet)/2);

%% Début boucle de calcul: filtrage et méthode TKE
    for i=1:(echant_canaux*paquet)/2

%% Retrouver l'amplitude
%Pour retrouver l'amplitude d'origine selon la plage de fonctionnement de l'ADC de
 l'ADS (on garde le gain de 12)
        data_in_canal_1(i)=(data_in_canal_1(i)*2.4)/(32767);

%% Filtrage passe-bande et coupe bande
%% Pour le canal 1

% Passe bande
        Y0_C1 = data_in_canal_1(i);

        Z0_C1 = NUM_BP(1)*Y0_C1 + NUM_BP(2)*Y1_C1 + NUM_BP(3)*Y2_C1 +
         NUM_BP(4)*Y3_C1 + NUM_BP(5)*Y4_C1 - (DEN_BP(2)*Z1_C1 + DEN_BP(3)*Z2_C1 +
         DEN_BP(4)*Z3_C1 + DEN_BP(5)*Z4_C1);
```

```
        Y4_C1 = Y3_C1; Y3_C1 = Y2_C1; Y2_C1 = Y1_C1; Y1_C1 = Y0_C1;
        Z4_C1 = Z3_C1; Z3_C1 = Z2_C1; Z2_C1 = Z1_C1; Z1_C1 = Z0_C1;

        data_in_canal_1(i) = Z0_C1;

% Coupe bande
        X0_C1 = data_in_canal_1(i);

        W0_C1 = NUM_BS(1)*X0_C1 + NUM_BS(2)*X1_C1 + NUM_BS(3)*X2_C1 -
         (DEN_BS(2)*W1_C1 + DEN_BS(3)*W2_C1);
        X2_C1 = X1_C1; X1_C1 = X0_C1;
        W2_C1 = W1_C1; W1_C1 = W0_C1;

% Données filtrées
        data_in_canal_1(i) = W0_C1;


%% TKE
        data_TKE_canal_1(i) = abs((data_in_canal_1(i).^2)-
         (data_in_canal_1(i+1)*xnm11));
        xnm11= data_in_canal_1(i);

%% Enregistrement des données intéressantes
        data_C1_load(indice) = data_in_canal_1(i);
        TKE_C1(indice) = data_TKE_canal_1(i);

        indice= indice+1;
    end


%% On détermine si il y a un mouvement ou non

     Cond1=(length(find(data_TKE_canal_1(1:end)<seuil))==length(data_TKE_canal_1(1
     :end)));

     Cond2=(length(find(data_TKE_canal_2(1:end)<seuil2))==length(data_TKE_canal_2(
     1:end)));

     Cond3=(length(find(data_TKE_canal_3(1:end)<seuil3))==length(data_TKE_canal_3(
     1:end)));

     Cond4=(length(find(data_TKE_canal_4(1:end)<seuil4))==length(data_TKE_canal_4(
     1:end)));

     Cond5=(length(find(data_TKE_canal_5(1:end)<seuil5))==length(data_TKE_canal_5(
     1:end)));

     Cond6=(length(find(data_TKE_canal_6(1:end)<seuil6))==length(data_TKE_canal_6(
     1:end)));

     Cond7=(length(find(data_TKE_canal_7(1:end)<seuil7))==length(data_TKE_canal_7(
     1:end)));

    taille_fenetre=192;
    taille_decalage_segment=16;
    nbre_segments=((taille_fenetre/taille_decalage_segment)-2);
```

```
    taille_segment=48;

    seuil_features=0.0005;
    freq_coupure=500;

% Si la condition est vraie, alors on est dans le cas d'un
% mouvement neutre (main ouverte) et on ne calcule pas les features
    if (Cond1 && Cond2 && Cond3 && Cond4 && Cond5 && Cond6 && Cond7)
        doigt_gagnant=1;

    else
% Sinon, on est dans le cas d'un mouvement : on calcule les features et on fait la
 prédiction
%% Calcul des features temporels pour la fenêtre en question

        for j=1:nbre_segments

            mav_C1(1,j)= mav(data_in_canal_1(1+
             (taille_decalage_segment*(j-1)):taille_fenetre-
             (taille_decalage_segment*(nbre_segments-j))));
            zero_crossing_C1(1,j)=zero_crossing(data_in_canal_1(1+
             (taille_decalage_segment*(j-1)):taille_fenetre-
             (taille_decalage_segment*(nbre_segments-j))), seuil_features);
            slope_signe_change_C1(1,j)=slope_sign_change(data_in_canal_1(1+
             (taille_decalage_segment*(j-1)):taille_fenetre-
             (taille_decalage_segment*(nbre_segments-j))), seuil_features);
            waveform_length_C1(1,j)=waveform_length(data_in_canal_1(1+
             (taille_decalage_segment*(j-1)):taille_fenetre-
             (taille_decalage_segment*(nbre_segments-j))));
            wamp_C1(1,j)=wilison_amplitude(data_in_canal_1(1+
             (taille_decalage_segment*(j-1)):taille_fenetre-
             (taille_decalage_segment*(nbre_segments-j))), seuil_features);
            AR_coeff_C1(1,(j-1)*4+1:(j-1)*4+4)=ar_yule_walker(data_in_canal_1(1+
             (taille_decalage_segment*(j-1)):taille_fenetre-
             (taille_decalage_segment*(nbre_segments-j))),4);

            skewness_C1(1,j)=skewness_value(data_in_canal_1(1+
             (taille_decalage_segment*(j-1)):taille_fenetre-
             (taille_decalage_segment*(nbre_segments-j))));
            iemg_C1(1,j)=iemg(data_in_canal_1(1+
             (taille_decalage_segment*(j-1)):taille_fenetre-
             (taille_decalage_segment*(nbre_segments-j))));
            hjorth_activity_C1(1,j)=hjorth_activity(data_in_canal_1(1+
             (taille_decalage_segment*(j-1)):taille_fenetre-
             (taille_decalage_segment*(nbre_segments-j))));
            hjorth_mobility_C1(1,j)=hjorth_mobility(data_in_canal_1(1+
             (taille_decalage_segment*(j-1)):taille_fenetre-
             (taille_decalage_segment*(nbre_segments-j))));
            hjorth_complexity_C1(1,j)=hjorth_complexity(data_in_canal_1(1+
             (taille_decalage_segment*(j-1)):taille_fenetre-
             (taille_decalage_segment*(nbre_segments-j))));


%% Calcul des features fréquentiels pour la fenêtre en question
% Calcul de la FFT
```

```
            Transform_C1=fftshift(fft(data_in_canal_1(1+
             (taille_decalage_segment*(j-1)):taille_fenetre-
             (taille_decalage_segment*(nbre_segments-j)))));
            Transform_C1_utile=Transform_C1(1+taille_segment/2:taille_segment);
            Power_spectral_density_C1=((abs(Transform_C1_utile)).^2)/freq_coupure;

            fe1=0:freq_coupure/(taille_segment/2-1):freq_coupure;

% Mean Frequency: MNF
            somme_spectre_C1=0;
            somme_produit_C1=0;

            for k=1:(taille_segment/2)
% Somme les amplis spectrales
                somme_spectre_C1 = somme_spectre_C1 +
                 Power_spectral_density_C1(k);
% Somme le produit des amplis spectrales et de la fréquence
                somme_produit_C1 = somme_produit_C1 +
                 (Power_spectral_density_C1(k)*fe1(k));
            end

            mean_freq_C1(1,j)=somme_produit_C1/somme_spectre_C1;
        end


%% Prédiction
        features_mesures=[mav_C1, zero_crossing_C1, slope_signe_change_C1,
         waveform_length_C1, wamp_C1, AR_coeff_C1, mean_freq_C1, skewness_C1,
         iemg_C1, hjorth_activity_C1, hjorth_mobility_C1, hjorth_complexity_C1,
         mav_C2, zero_crossing_C2, slope_signe_change_C2, waveform_length_C2,
         wamp_C2, AR_coeff_C2, mean_freq_C2, skewness_C2, iemg_C2,
         hjorth_activity_C2, hjorth_mobility_C2, hjorth_complexity_C2, mav_C3,
         zero_crossing_C3, slope_signe_change_C3, waveform_length_C3, wamp_C3,
         AR_coeff_C3, mean_freq_C3, skewness_C3, iemg_C3, hjorth_activity_C3,
         hjorth_mobility_C3, hjorth_complexity_C3, mav_C4, zero_crossing_C4,
         slope_signe_change_C4, waveform_length_C4, wamp_C4, AR_coeff_C4,
         mean_freq_C4, skewness_C4, iemg_C4, hjorth_activity_C4,
         hjorth_mobility_C4, hjorth_complexity_C4, mav_C5, zero_crossing_C5,
         slope_signe_change_C5, waveform_length_C5, wamp_C5, AR_coeff_C5,
         mean_freq_C5, skewness_C5, iemg_C5, hjorth_activity_C5,
         hjorth_mobility_C5, hjorth_complexity_C5, mav_C6, zero_crossing_C6,
         slope_signe_change_C6, waveform_length_C6, wamp_C6, AR_coeff_C6,
         mean_freq_C6, skewness_C6, iemg_C6, hjorth_activity_C6,
         hjorth_mobility_C6, hjorth_complexity_C6, mav_C7, zero_crossing_C7,
         slope_signe_change_C7, waveform_length_C7, wamp_C7, AR_coeff_C7,
         mean_freq_C7, skewness_C7, iemg_C7, hjorth_activity_C7,
         hjorth_mobility_C7, hjorth_complexity_C7];
        doigt_gagnant=predict(Mdl,features_mesures);

    end


%% Affichage du doigt gagnant (toutes les 192ms)
    if (doigt_gagnant==1)
        set(handles.Prediction_canaux,'String','1');
        set(handles.text_info,'String','Main ouverte','ForegroundColor',[0 ,0 ,
         1],'FontSize',60);
```

```
    imh = imhandles(handles.axes8);
    set(imh,'CData',img_main_ouverte,'XData',[0 500],'YData',[0 500]);

elseif (doigt_gagnant==2)
    set(handles.Prediction_canaux,'String','2');
    set(handles.text_info,'String','Flexion Pouce','ForegroundColor',[0 ,0 ,
     1],'FontSize',60);
    imh = imhandles(handles.axes8);
    set(imh,'CData',img_pouce,'XData',[0 500],'YData',[0 500]);

elseif (doigt_gagnant==3)
    set(handles.Prediction_canaux,'String','3');
    set(handles.text_info,'String','Flexion Index','ForegroundColor',[0 ,0 ,
     1],'FontSize',60);
    imh = imhandles(handles.axes8);
    set(imh,'CData',img_index,'XData',[0 500],'YData',[0 500]);

elseif (doigt_gagnant==4)
    set(handles.Prediction_canaux,'String','4');
    set(handles.text_info,'String','Flexion Majeur','ForegroundColor',[0 ,0 ,
     1],'FontSize',60);
    imh = imhandles(handles.axes8);
    set(imh,'CData',img_majeur,'XData',[0 500],'YData',[0 500]);

elseif (doigt_gagnant==5)
    set(handles.Prediction_canaux,'String','5');
    set(handles.text_info,'String','Flexion Annulaire','ForegroundColor',[0 ,0
     ,1],'FontSize',60);
    imh = imhandles(handles.axes8);
    set(imh,'CData',img_annulaire,'XData',[0 500],'YData',[0 500]);

elseif (doigt_gagnant==6)
    set(handles.Prediction_canaux,'String','6');
    set(handles.text_info,'String','Flexion Auriculaire','ForegroundColor',
     [0 ,0 ,1],'FontSize',60);
    imh = imhandles(handles.axes8);
    set(imh,'CData',img_auriculaire,'XData',[0 500],'YData',[0 500]);

elseif (doigt_gagnant==7)
    set(handles.Prediction_canaux,'String','7');
    set(handles.text_info,'String','Pince simple','ForegroundColor',[0 ,0 ,
     1],'FontSize',60);
    imh = imhandles(handles.axes8);
    set(imh,'CData',img_pince_simple,'XData',[0 500],'YData',[0 500]);

elseif (doigt_gagnant==8)
    set(handles.Prediction_canaux,'String','8');
    set(handles.text_info,'String','Pince complexe','ForegroundColor',[0 ,0 ,
     1],'FontSize',60);
    imh = imhandles(handles.axes8);
    set(imh,'CData',img_pince_complexe,'XData',[0 500],'YData',[0 500]);

elseif (doigt_gagnant==9)
    set(handles.Prediction_canaux,'String','9');
    set(handles.text_info,'String','Pointer index','ForegroundColor',[0 ,0 ,
     1],'FontSize',60);
    imh = imhandles(handles.axes8);
```

```
        set(imh,'CData',img_pointer_index,'XData',[0 500],'YData',[0 500]);

    elseif (doigt_gagnant==10)
        set(handles.Prediction_canaux,'String','10');
        set(handles.text_info,'String','Flexion 3 doigts','ForegroundColor',[0 ,
         0 ,1],'FontSize',60);
        imh = imhandles(handles.axes8);
        set(imh,'CData',img_flexion_3doigts,'XData',[0 500],'YData',[0 500]);

    elseif (doigt_gagnant==11)
        set(handles.Prediction_canaux,'String','11');
        set(handles.text_info,'String','Main fermée','ForegroundColor',[0 ,0 ,
         1],'FontSize',60);
        imh = imhandles(handles.axes8);
        set(imh,'CData',img_main_fermee,'XData',[0 500],'YData',[0 500]);

    elseif (doigt_gagnant==12)
        set(handles.Prediction_canaux,'String','12');
        set(handles.text_info,'String','Prise cylindrique','ForegroundColor',[0 ,0
         ,1],'FontSize',60);
        imh = imhandles(handles.axes8);
        set(imh,'CData',img_prise_cylindrique,'XData',[0 500],'YData',[0 500]);

    elseif (doigt_gagnant==13)
        set(handles.Prediction_canaux,'String','13');
        set(handles.text_info,'String','Prise latérale','ForegroundColor',[0 ,0 ,
         1],'FontSize',60);
        imh = imhandles(handles.axes8);
        set(imh,'CData',img_prise_laterale,'XData',[0 500],'YData',[0 500]);

    end

%% Réinitialisation des variables
    mean_freq_C1=zeros(1,length(nbre_segments));
    mav_C1=zeros(1,length(nbre_segments));
    zero_crossing_C1=zeros(1,length(nbre_segments));
    slope_signe_change_C1=zeros(1,length(nbre_segments));
    waveform_length_C1=zeros(1,length(nbre_segments));
    wamp_C1=zeros(1,length(nbre_segments));
    AR_coeff_C1=zeros(1,4*length(nbre_segments));
    skewness_C1=zeros(1,length(nbre_segments));
    iemg_C1=zeros(1,length(nbre_segments));
    hjorth_activity_C1=zeros(1,length(nbre_segments));
    hjorth_mobility_C1=zeros(1,length(nbre_segments));
    hjorth_complexity_C1=zeros(1,length(nbre_segments));
```

Initialisations pour l'interface de prédiction en temps réel :

```
% --- Executes just before Interface_LDA is made visible.
    function Interface_LDA_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Interface_LDA (see VARARGIN)

    global xnm11
    xnm11=0;

    global Y1_C1 Y2_C1 Y3_C1 Y4_C1 Z1_C1 Z2_C1 Z3_C1 Z4_C1
    Y1_C1=0; Y2_C1=0; Y3_C1=0; Y4_C1=0; Z1_C1=0; Z2_C1=0; Z3_C1=0; Z4_C1=0;

    global W1_C1 W2_C1 X1_C1 X2_C1
    W1_C1=0; W2_C1=0; X1_C1=0; X2_C1=0;

    global indice
    indice= 1;

    global board
    global temps

    global freq_echant
    global ScopeWidth Nb_curves

    global NUM_BP DEN_BP
    global NUM_BS DEN_BS

    global flag
    flag=0;

    global echant_canaux paquet nb_canaux taille_paquet
% 64 moitiés d'échantillons/canal donc 32 échantillons en tout
    echant_canaux=64;
    paquet=6;
    nb_canaux=7;
    taille_paquet=(nb_canaux*echant_canaux)+2;

    global XB_bytes
    global NB_bytes_to_read

    NB_bytes_to_read = (((echant_canaux*nb_canaux)+2)*paquet);
% 2700 car paquet de 64X7 données intéressantes (car échantillon en 2 parties donc
 32X2),
% un 0 de départ et un 1 de fin. On attend d'avoir 6 paquets avant de lire le
 buffer du PC
    XB_bytes = NB_bytes_to_read;

    global seuil
    seuil=0;

    global img_pouce img_index img_majeur img_annulaire img_auriculaire
     img_flexion_3doigts
    global img_pointer_index img_pince_simple img_pince_complexe
     img_prise_cylindrique img_prise_laterale
    global img_main_fermee img_main_ouverte
```

```
% Coupe bande d'ordre 2
    NUM_BS(1)=0.842646292224708;
    NUM_BS(2)=-1.557390190740875;
    NUM_BS(3)=0.832369385605689;
    DEN_BS(1)=1;
    DEN_BS(2)=-1.557390190740875;
    DEN_BS(3)=0.675015677830398;

% Filtrage passe bande avec un Butterworth
    w1 = 20/(freq_echant/2);
    w2 = 490/(freq_echant/2);
    [NUM_BP,DEN_BP]=butter(2,[w1,w2],'bandpass');

    delete (instrfindall);

% Create a COM Port
    board = serial('/dev/tty.SLAB_USBtoUART', 'BaudRate', 921600, 'DataBits',8);
    set(board,'InputBufferSize', 100000);
    fopen(board);

% Initialize the stripchart
% Sampling frequency
    freq_echant = 1000;
% Defines the stripchart's scope width
    ScopeWidth = 1;
% Number of curves plotted in the stripchart
    Nb_curves = 1;

    stripchart(handles.axes1, freq_echant, ScopeWidth, Nb_curves);
    ylabel(handles.axes1,'Tension (V)') ;
    title(handles.axes1,'Signal EMG du canal 1');

    ylim(handles.axes1,[-0.005 0.005]);

% Setup the timer
    TMR_PERIOD = 0.005;
    temps = timer('TimerFcn', @(x,y)routine(hObject, eventdata, handles),
     'Period', TMR_PERIOD);
    set(temps,'ExecutionMode','fixedRate');

% Charge les images JPG
    img_pouce=imread('flexion_pouce.png');
    img_index=imread('flexion_index.png');
    img_majeur=imread('flexion_majeur.png');
    img_annulaire=imread('flexion_annulaire.png');
    img_auriculaire=imread('flexion_auriculaire.png');
    img_flexion_3doigts=imread('flexion_3doigts.png');
    img_main_ouverte=imread('main_ouverte.png');
    img_main_fermee=imread('main_fermee.png');
    img_pointer_index=imread('pointer_index.png');
    img_pince_simple=imread('pince_simple.png');
    img_pince_complexe=imread('pince_complexe.png');
    img_prise_cylindrique=imread('prise_cylindrique.png');
    img_prise_laterale=imread('prise_laterale.png');

    handles.axes8=image(img_main_ouverte,'XData',[0 500],'YData',[0 500]);
```

```
% Choose default command line output for Interface_LDA
    handles.output = hObject;

% Update handles structure
    guidata(hObject, handles);
```

Chargement de la base de données pour l'interface de prédiction en temps réel :

```
% --- Executes on button press in Bouton_charger_database.
    function Bouton_charger_database_Callback(hObject, eventdata, handles)
% hObject    handle to Bouton_charger_database (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% dossier correspondra au nom du dossier sélectionné dans le menu déroulant
    global dossier
    global Main_ouverte Flexion_Pouce Flexion_Index Flexion_Majeur
     Flexion_Annulaire Flexion_Auriculaire Main_fermee Pince_simple Pince_complexe
     Pointer_index Flexion_3_doigts Prise_cylindrique Prise_laterale
    global Mdl
    global Features
    global Classes

    Main_ouverte=1;
    Flexion_Pouce=2;
    Flexion_Index=3;
    Flexion_Majeur=4;
    Flexion_Annulaire=5;
    Flexion_Auriculaire=6;
    Pince_simple=7;
    Pince_complexe=8;
    Pointer_index=9;
    Flexion_3_doigts=10;
    Main_fermee=11;
    Prise_cylindrique=12;
    Prise_laterale=13;

    taille_fenetre=192;
    nbre_sample_enregistre=5120;
    taille_decalage=64;
    nbre_fenetres=((nbre_sample_enregistre/taille_decalage)-2);

    taille_decalage_segment=16;
    nbre_segments=((taille_fenetre/taille_decalage_segment)-2);

% Nombre d'observations : on fait juste 10 enregistrements/classe
    No=10;
% Nombre de features
    Nf=15;

    Features=zeros(No*13*nbre_fenetres,Nf*7*nbre_segments);
% 13 = nombre de classes, Nf*7 car 7 canaux
    Classes=zeros(No*13*nbre_fenetres,1);

    for i=1:No

% Mvt 1
        load(strcat(dossier{1},'/Features/
         Main_ouverte_Neutre_obs_',num2str(i),'.mat'));

% Construire la matrice features et le vecteur classe
        Features((1+(i-1)*nbre_fenetres):(i*nbre_fenetres),:)=[canal_1_features,
         canal_2_features, canal_3_features, canal_4_features, canal_5_features,
         canal_6_features, canal_7_features];
        Classes((1+(i-1)*nbre_fenetres):(i*nbre_fenetres),1)=Main_ouverte;
```

```
% Mvt 2
        load(strcat(dossier{1},'/Features/Flexion_Pouce_obs_',num2str(i),'.mat'));

        Features((1+(i-1)*nbre_fenetres+(No*nbre_fenetres)):(i*nbre_fenetres+
         (No*nbre_fenetres)),:)=[canal_1_features, canal_2_features,
         canal_3_features, canal_4_features, canal_5_features, canal_6_features,
         canal_7_features];
        Classes((1+(i-1)*nbre_fenetres+(No*nbre_fenetres)):(i*nbre_fenetres+
         (No*nbre_fenetres)),1)=Flexion_Pouce;

% Mvt 3
        load(strcat(dossier{1},'/Features/Flexion_Index_obs_',num2str(i),'.mat'));

        Features((1+(i-1)*nbre_fenetres+(2*No*nbre_fenetres)):
         (i*nbre_fenetres+(2*No*nbre_fenetres)),:)=[canal_1_features,
         canal_2_features, canal_3_features, canal_4_features, canal_5_features,
         canal_6_features, canal_7_features];
        Classes((1+(i-1)*nbre_fenetres+(2*No*nbre_fenetres)):
         (i*nbre_fenetres+(2*No*nbre_fenetres)),1)=Flexion_Index;

% Mvt 4
        load(strcat(dossier{1},'/Features/
         Flexion_Majeur_obs_',num2str(i),'.mat'));

        Features((1+(i-1)*nbre_fenetres+(3*No*nbre_fenetres)):
         (i*nbre_fenetres+(3*No*nbre_fenetres)),:)=[canal_1_features,
         canal_2_features, canal_3_features, canal_4_features, canal_5_features,
         canal_6_features, canal_7_features];
        Classes((1+(i-1)*nbre_fenetres+(3*No*nbre_fenetres)):
         (i*nbre_fenetres+(3*No*nbre_fenetres)),1)=Flexion_Majeur;

% Mvt 5
        load(strcat(dossier{1},'/Features/
         Flexion_Annulaire_obs_',num2str(i),'.mat'));

        Features((1+(i-1)*nbre_fenetres+(4*No*nbre_fenetres)):
         (i*nbre_fenetres+(4*No*nbre_fenetres)),:)=[canal_1_features,
         canal_2_features, canal_3_features, canal_4_features, canal_5_features,
         canal_6_features, canal_7_features];
        Classes((1+(i-1)*nbre_fenetres+(4*No*nbre_fenetres)):
         (i*nbre_fenetres+(4*No*nbre_fenetres)),1)=Flexion_Annulaire;

% Mvt 6
        load(strcat(dossier{1},'/Features/
         Flexion_Auriculaire_obs_',num2str(i),'.mat'));

        Features((1+(i-1)*nbre_fenetres+(5*No*nbre_fenetres)):
         (i*nbre_fenetres+(5*No*nbre_fenetres)),:)=[canal_1_features,
         canal_2_features, canal_3_features, canal_4_features, canal_5_features,
         canal_6_features, canal_7_features];
        Classes((1+(i-1)*nbre_fenetres+(5*No*nbre_fenetres)):
         (i*nbre_fenetres+(5*No*nbre_fenetres)),1)=Flexion_Auriculaire;

% Mvt 7
        load(strcat(dossier{1},'/Features/
         Pince_simple_Pince_obs_',num2str(i),'.mat'));
```

```
        Features((1+(i-1)*nbre_fenetres+(6*No*nbre_fenetres)):
         (i*nbre_fenetres+(6*No*nbre_fenetres)),:)=[canal_1_features,
         canal_2_features, canal_3_features, canal_4_features, canal_5_features,
         canal_6_features, canal_7_features];
        Classes((1+(i-1)*nbre_fenetres+(6*No*nbre_fenetres)):
         (i*nbre_fenetres+(6*No*nbre_fenetres)),1)=Pince_simple;

% Mvt 8
        load(strcat(dossier{1},'/Features/
         Pince_complexe_Pince_obs_',num2str(i),'.mat'));

        Features((1+(i-1)*nbre_fenetres+(7*No*nbre_fenetres)):
         (i*nbre_fenetres+(7*No*nbre_fenetres)),:)=[canal_1_features,
         canal_2_features, canal_3_features, canal_4_features, canal_5_features,
         canal_6_features, canal_7_features];
        Classes((1+(i-1)*nbre_fenetres+(7*No*nbre_fenetres)):
         (i*nbre_fenetres+(7*No*nbre_fenetres)),1)=Pince_complexe;

% Mvt 9
        load(strcat(dossier{1},'/Features/
         Pointer_index_Pointer_obs_',num2str(i),'.mat'));

        Features((1+(i-1)*nbre_fenetres+(8*No*nbre_fenetres)):
         (i*nbre_fenetres+(8*No*nbre_fenetres)),:)=[canal_1_features,
         canal_2_features, canal_3_features, canal_4_features, canal_5_features,
         canal_6_features, canal_7_features];
        Classes((1+(i-1)*nbre_fenetres+(8*No*nbre_fenetres)):
         (i*nbre_fenetres+(8*No*nbre_fenetres)),1)=Pointer_index;

% Mvt 10
        load(strcat(dossier{1},'/Features/
         Flexion_3_doigts_obs_',num2str(i),'.mat'));

        Features((1+(i-1)*nbre_fenetres+(9*No*nbre_fenetres)):
         (i*nbre_fenetres+(9*No*nbre_fenetres)),:)=[canal_1_features,
         canal_2_features, canal_3_features, canal_4_features, canal_5_features,
         canal_6_features, canal_7_features];
        Classes((1+(i-1)*nbre_fenetres+(9*No*nbre_fenetres)):
         (i*nbre_fenetres+(9*No*nbre_fenetres)),1)=Flexion_3_doigts;

% Mvt 11
        load(strcat(dossier{1},'/Features/
         Main_fermee_Fermer_obs_',num2str(i),'.mat'));

        Features((1+(i-1)*nbre_fenetres+(10*No*nbre_fenetres)):
         (i*nbre_fenetres+(10*No*nbre_fenetres)),:)=[canal_1_features,
         canal_2_features, canal_3_features, canal_4_features, canal_5_features,
         canal_6_features, canal_7_features];
        Classes((1+(i-1)*nbre_fenetres+(10*No*nbre_fenetres)):
         (i*nbre_fenetres+(10*No*nbre_fenetres)),1)=Main_fermee;

% Mvt 12
        load(strcat(dossier{1},'/Features/
         Prise_cylindrique_Cylindrique_obs_',num2str(i),'.mat'));
```

```
        Features((1+(i-1)*nbre_fenetres+(11*No*nbre_fenetres)):
         (i*nbre_fenetres+(11*No*nbre_fenetres)),:)=[canal_1_features,
         canal_2_features, canal_3_features, canal_4_features, canal_5_features,
         canal_6_features, canal_7_features];
        Classes((1+(i-1)*nbre_fenetres+(11*No*nbre_fenetres)):
         (i*nbre_fenetres+(11*No*nbre_fenetres)),1)=Prise_cylindrique;

% Mvt 13
        load(strcat(dossier{1},'/Features/
         Prise_laterale_Laterale_obs_',num2str(i),'.mat'));

        Features((1+(i-1)*nbre_fenetres+(12*No*nbre_fenetres)):
         (i*nbre_fenetres+(12*No*nbre_fenetres)),:)=[canal_1_features,
         canal_2_features, canal_3_features, canal_4_features, canal_5_features,
         canal_6_features, canal_7_features];
        Classes((1+(i-1)*nbre_fenetres+(12*No*nbre_fenetres)):
         (i*nbre_fenetres+(12*No*nbre_fenetres)),1)=Prise_laterale;

    end

% Classification
    Mdl=fitcdiscr(Features,Classes,'discrimType','pseudoLinear');

    set(handles.text_info,'String','Databases des canaux
     chargées','ForegroundColor',[1 ,0 ,0],'FontSize',20);
```

Calibration de la main ouverte pour l'interface de prédiction en temps réel :

```
% --- Executes on button press in Calibration_neutre.
    function Calibration_neutre_Callback(hObject, eventdata, handles)
% hObject    handle to Calibration_neutre (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

    global board
    global temps
    global indice
    global donnees
    global flag
    global XB_bytes
    global NB_bytes_to_read

    global data_C1_load
    global TKE_C1

    global seuil

    data_C1_load=[];
    TKE_C1=[];
    XB_bytes=NB_bytes_to_read;

    indice=1;

    stop(temps);

% Lit les données et les supprime après
    donnees = fread(board, board.BytesAvailable);

    flag=0;

    set(handles.text_info,'String','Calibration en cours','ForegroundColor',[0 ,
     1 ,0.5],'FontSize',20);
    pause(1);

    start(temps);
    pause(5);
    stop(temps);

% Le max des artefacts est le seuil
    seuil=max(TKE_C1(1:end));

    set(handles.text_info,'String','Fin de la Calibration','ForegroundColor',[0 ,1
     ,0.5],'FontSize',20);
    pause(1);

    start(temps);
```

Calculs des paramètres discriminants pour l'interface de construction de la base de données :

```
% --- Executes on button press in Bouton_features.
    function Bouton_features_Callback(hObject, eventdata, handles)
% hObject    handle to Bouton_features (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

    global data_C1_load
    global data_C1_plage_mvt
    global TKE_C1

    global mean_freq_C1
    global mav_C1
    global zero_crossing_C1
    global slope_signe_change_C1
    global waveform_length_C1
    global wamp_C1
    global AR_coeff_C1
    global skewness_C1
    global iemg_C1
    global hjorth_activity_C1
    global hjorth_mobility_C1
    global hjorth_complexity_C1

% Dépend du choix du mouvement dans le menu déroulant de l'interface
    global mvt_stv

% On établit un seuil pour voir à quel moment le mouvement commence et
% on prend 5s d'echantillons à partir de ce moment
    moy_TKE_C1=mean(TKE_C1(100:end));
% A partir de 100 pour éviter l'instabilité du coupe bande
    ecart_type_TKE_C1=std(TKE_C1(100:end));
    seuil=moy_TKE_C1+(5*ecart_type_TKE_C1);

    u=1;
    flag=0;
    nbre_sample_enregistre=5120;

%% Enregistrement plage 5s dans le cas de mvt
    if (mvt_stv==2 || mvt_stv==3 || mvt_stv==4 || mvt_stv==5 || mvt_stv==7 ||
     mvt_stv==8 || mvt_stv==9)
        for i=100:length(data_C1_load)

            if ((TKE_C1(i)>=seuil || TKE_C2(i)>=seuil2 || TKE_C3(i)>=seuil3 ||
             TKE_C4(i)>=seuil4 || TKE_C5(i)>=seuil5 || TKE_C6(i)>=seuil6 ||
             TKE_C7(i)>=seuil7) && u<=nbre_sample_enregistre)
% Lorsqu'on passe au dessus de l'un des seuils pour la première fois, on
 enregistre les données
                data_C1_plage_mvt(u)=data_C1_load(i);
                u=u+1;
% On lève le flag pour dire que le mvt a commencé
                flag=1;

            elseif (u<=nbre_sample_enregistre && flag==1)
% Lorsqu'on est en dessous ou au dessus du seuil mais que l'on
% n'a pas encore enregistré 5s, on continue d'enregistrer les données
                data_C1_plage_mvt(u)=data_C1_load(i);
                u=u+1;
```

```
            elseif (u>nbre_sample_enregistre || flag==0)
% Lorsqu'on a pris 5s d'échantillons ou que le mvt n'a pas commencé, on ne fait
 rien
                u=u;
            end
        end
    end

% Dans le cas d'un mvt neutre (main ouverte) (pas de seuil)
    if (mvt_stv==6)
        for i=2000:7119
% On enregistre les données
            data_C1_plage_mvt(u)=data_C1_load(i);
            u=u+1;
        end
    end

%% Calcul des features
% Découpage en fenêtres overlapées de 128ms (64ms de décalage à chaque fois)
% 78 fenêtres donc 78 observations pour un enregistrement de mouvement de 5s
% Chaque fenêtre est coupée en 10 segments sur lesquels on calcule
% les features. On a toujours 78 observations pour un
% enregistrement mais 10* le nbre de feature initial

    taille_fenetre=192;
    taille_decalage=64;
    nbre_fenetres=((nbre_sample_enregistre/taille_decalage)-2);

    taille_segment=48;
    taille_decalage_segment=16;
    nbre_segments=((taille_fenetre/taille_decalage_segment)-2);

    seuil_features=0.0005;
    freq_coupure=500;

    mav_C1=zeros(length(nbre_fenetres),length(nbre_segments));
    zero_crossing_C1=zeros(length(nbre_fenetres),length(nbre_segments));
    slope_signe_change_C1=zeros(length(nbre_fenetres),length(nbre_segments));
    waveform_length_C1=zeros(length(nbre_fenetres),length(nbre_segments));
    wamp_C1=zeros(length(nbre_fenetres),length(nbre_segments));
    AR_coeff_C1=zeros(length(nbre_fenetres),4*length(nbre_segments));
    skewness_C1=zeros(length(nbre_fenetres),length(nbre_segments));
    iemg_C1=zeros(length(nbre_fenetres),length(nbre_segments));
    hjorth_activity_C1=zeros(length(nbre_fenetres),length(nbre_segments));
    hjorth_mobility_C1=zeros(length(nbre_fenetres),length(nbre_segments));
    hjorth_complexity_C1=zeros(length(nbre_fenetres),length(nbre_segments));
    mean_freq_C1=zeros(length(nbre_fenetres),length(nbre_segments));

% Nombre de fenêtres overlapées
    for i=1:nbre_fenetres

        for j=1:nbre_segments
```

```
                mav_C1(i,j)= mav(data_C1_plage_mvt(((taille_decalage*(i-1))+1)+
                 (taille_decalage_segment*(j-1)):taille_fenetre+
                 (taille_decalage*(i-1))-(taille_decalage_segment*(nbre_segments-
                 j))));

                 zero_crossing_C1(i,j)=zero_crossing(data_C1_plage_mvt(((taille_decala
                 ge*(i-1))+1)+(taille_decalage_segment*(j-1)):taille_fenetre+
                 (taille_decalage*(i-1))-(taille_decalage_segment*(nbre_segments-
                 j))),seuil_features);

                 slope_signe_change_C1(i,j)=slope_sign_change(data_C1_plage_mvt(((tail
                 le_decalage*(i-1))+1)+(taille_decalage_segment*(j-1)):taille_fenetre+
                 (taille_decalage*(i-1))-(taille_decalage_segment*(nbre_segments-j))),
                 seuil_features);

                 waveform_length_C1(i,j)=waveform_length(data_C1_plage_mvt(((taille_de
                 calage*(i-1))+1)+(taille_decalage_segment*(j-1)):taille_fenetre+
                 (taille_decalage*(i-1))-(taille_decalage_segment*(nbre_segments-
                 j))));

                 wamp_C1(i,j)=wilison_amplitude(data_C1_plage_mvt(((taille_decalage*(i
                 -1))+1)+(taille_decalage_segment*(j-1)):taille_fenetre+
                 (taille_decalage*(i-1))-(taille_decalage_segment*(nbre_segments-j))),
                 seuil_features);
                AR_coeff_C1(i,(j-1)*4+1:
                 (j-1)*4+4)=ar_yule_walker(data_C1_plage_mvt(((taille_decalage*(i-1))
                 +1)+(taille_decalage_segment*(j-1)):taille_fenetre+
                 (taille_decalage*(i-1))-(taille_decalage_segment*(nbre_segments-j))),
                 4);


                 skewness_C1(i,j)=skewness_value(data_C1_plage_mvt(((taille_decalage*(
                 i-1))+1)+(taille_decalage_segment*(j-1)):taille_fenetre+
                 (taille_decalage*(i-1))-(taille_decalage_segment*(nbre_segments-
                 j))));
                iemg_C1(i,j)=iemg(data_C1_plage_mvt(((taille_decalage*(i-1))+1)+
                 (taille_decalage_segment*(j-1)):taille_fenetre+
                 (taille_decalage*(i-1))-(taille_decalage_segment*(nbre_segments-
                 j))));

                 hjorth_activity_C1(i,j)=hjorth_activity(data_C1_plage_mvt(((taille_de
                 calage*(i-1))+1)+(taille_decalage_segment*(j-1)):taille_fenetre+
                 (taille_decalage*(i-1))-(taille_decalage_segment*(nbre_segments-
                 j))));

                 hjorth_mobility_C1(i,j)=hjorth_mobility(data_C1_plage_mvt(((taille_de
                 calage*(i-1))+1)+(taille_decalage_segment*(j-1)):taille_fenetre+
                 (taille_decalage*(i-1))-(taille_decalage_segment*(nbre_segments-
                 j))));

                 hjorth_complexity_C1(i,j)=hjorth_complexity(data_C1_plage_mvt(((taill
                 e_decalage*(i-1))+1)+(taille_decalage_segment*(j-1)):taille_fenetre+
                 (taille_decalage*(i-1))-(taille_decalage_segment*(nbre_segments-
                 j))));

% Calcul du feature fréquentiel
% Calcul de la FFT
```

```
            Transform_C1=fftshift(fft(data_C1_plage_mvt(((taille_decalage*(i-1))
             +1)+(taille_decalage_segment*(j-1)):taille_fenetre+
             (taille_decalage*(i-1))-(taille_decalage_segment*(nbre_segments-
             j)))));
            Transform_C1_utile=Transform_C1(1+taille_segment/2:taille_segment);
            Power_spectral_density_C1=((abs(Transform_C1_utile)).^2)/freq_coupure;

            fe1=0:freq_coupure/(taille_segment/2-1):freq_coupure;

% Mean Frequency: MNF
            somme_spectre_C1=0;
            somme_produit_C1=0;

            for k=1:(taille_segment/2)
% Somme les amplis spectrales
                somme_spectre_C1 = somme_spectre_C1 +
                 Power_spectral_density_C1(k);
% Somme le produit des amplis spectrales et de la fréquence
                somme_produit_C1 = somme_produit_C1 +
                 (Power_spectral_density_C1(k)*fe1(k));
            end

            mean_freq_C1(i,j)=somme_produit_C1/somme_spectre_C1;
        end
    end

    set(handles.text_info,'String','Calcul des features
     effectué','ForegroundColor',[1 ,0 ,0],'FontSize',20);
```

Annexe B : Code C implémenté dans le microcontrôleur MSP430F5529 (Partie 5)

Main.cpp :

```
#include <msp430.h>
#include <stdio.h>
#include <string.h>

#include "afe.h"
#include "driverlib.h"
#include "kernelcfg.h"
#include "utils.h"
#include "types.h"

int flag = 0; // Variable utilisée pour dire qu'on est prêt à envoyer les données
 par UART

// On écrit de façon alternée dans le buffer 1 puis 2. Pendant qu'on écrit dans le
 2 en interruption, on a le temps de finir de lire le 1 par UART
volatile int buffer1[450];
volatile int buffer2[450];
volatile int * ptrlecture = buffer1; // Pointeur utilisé dans le main (pour UART)
 sur le tableau qui contient les données
volatile int * ptrecriture = buffer1; // Pointeur utilisé dans l'interruption sur
 le tableau qui contient les données
volatile int double_buffer_id = 0; // Variable utilisée pour switcher entre les
 buffers 1 et 2

char toggle=0; // Va osciller entre 0 et 1 et permettre de prendre les données SPI
 une fois sur 2: pour une fréq d'échantillonage de 1kHz


void main(void)
{
    int TX_data_buffer[450]; // Variable à transmettre par UART avec les 32
     échantillons/canaux (2bytes=1échant) + un 254 + un 1 avec freq
     échantillonnage à 1kHz
    int nb_echant_canaux = 32;

    // Variables utilisées dans les boucles pour l'envoie par UART
    static int i = 0;
    static int j = 0;
    static int nb = 0;

    WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer : un timer spécial qui
     réinitialise le microcontrôleur après un certain temps pour éviter les
     boucles infinies


    //-------------------------------------------------------------------------
    // Structure pour changer la fréquence d'horloge du microcontrôleur (8MHz)
    //-------------------------------------------------------------------------

    UCSCTL3 = 0x20;
    UCSCTL4 |= SELA_2;
    UCSCTL0 = 0x0;

    do
    {
            UCSCTL7 &= ~(BIT2);
            UCSCTL7 &= ~(BIT3 | BIT1 | BIT0);
```

```
        SFRIFG1 &= ~BIT1;

}while(SFRIFG1&OFIFG); // Permet de nettoyer les drapeaux du registre SFRIFG1

__bis_SR_register(SCG0); // Désactive la boucle de contrôle FLL

UCSCTL1 |= (BIT6 | BIT5);
UCSCTL1 &= ~(BIT4| BIT0);

UCSCTL2 |= (BIT7 | BIT6 | BIT5 | BIT4 | BIT1 | BIT0); // Fixe la valeur de la
 fréquence
UCSCTL2 &= ~(BIT9 | BIT8 | BIT3 | BIT2);

__bic_SR_register(SCG0); // Active la boucle de contrôle FLL

// Le pin P7.7 est le signal de sortie de l'horloge
P7DIR |= BIT7;  // Initialise le pin P7.7 en sortie
               // PxDIR est le registre qui permet de définir la direction des
                pins GPIOx
P7SEL |= BIT7; // Alternate function

//-------------------------------------------------------------------------------
// Fin de la structure pour la configuration de la fréquence d'horloge
//-------------------------------------------------------------------------------


//-------------------------------------------------------------------------------
// UART setup USCI_A1
//-------------------------------------------------------------------------------

// Les pins P4.4 (TX) et P4.5 (RX) permettent de transmettre/recevoir les
 données
P4DIR &= ~BIT4;
P4DIR &= ~BIT5;
P4SEL |= BIT4;
P4SEL |= BIT5;

// Reset l'UART
UCA1CTL1 |= BIT0;

// Choisit la clock utilisée : SMCLK
UCA1CTL1 |= (BIT7);
UCA1CTL1 &= ~(BIT6);

// Impose le BaudRate à 921600 bps
UCA1BR0 = 0x09;
UCA1BR1 = 0x00;

// Pour démarrer l'UART
UCA1CTL1 &= ~BIT0;

// Active l'interruption sur RX à chaque réception de bits
UCA1IE |= BIT0;

//-------------------------------------------------------------------------------
// Fin de l'UART setup
//-------------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------
// Setup SPI entre le MSP et l'ADS, AFE_CLK est l'horloge du SPI
//-------------------------------------------------------------------------

__delay_cycles(2000000); // Pour laisser le temps au système de faire ses
 configurations avant d'envoyer les commandes SPI

// Pour mettre à 1 CLKSEL : permet de lancer l'horloge interne de l'ADS pour
 sa partie numérique
P1DIR |= BIT7;
P1OUT |= BIT7;

// Pour activer le pin P2.2 comme une sortie, qu'il puisse renvoyer la
 fréquence d'horloge du microcontrôleur (SMCLK=MCLK=8MHz)
P2DIR |= BIT2;
P2SEL |= BIT2;

AFE_setupGPIO(); // Définit les ports de connections entre le MSP et l'ADS
AFE_setupSPI(SPI_HIGH_SPEED); // On configure le SPI à 8MHz

__delay_cycles(500000);

AFE_PWR_UP; // Ici, on réveille le système
AFE_RESET_RELEASE; // Reset relaché

__delay_cycles(2000000);

AFE_hardReset(); // Effectue un Reset

__delay_cycles(2000000);

AFE_sendCmd(AFE_CMD_SDATAC); // STOP DATA ACQUISITION

AFE_writeRegister(AFE_REG_CONFIG3, 0xC0); // Pour écrire une valeur dans un
 registre avec le SPI. Here: Enable internal reference buffer

__delay_cycles(2000000);

AFE_initIRQ(); // Pour activer les interruptions (flag) sur nDRDY et le
 configurer comme entrée

__delay_cycles(40);

AFE_writeRegister(AFE_REG_CONFIG1, 0x23); // Low-Power and 2 kSPS, BP=524Hz

AFE_writeRegister(AFE_REG_CONFIG3, 0xCC); // ADC internal reference of 2.4V
 and RLD on with lead-off disabled

AFE_writeRegister(AFE_REG_CH1SET, 0x60); // Normal operation on CH1 and
 gain=12, normal electrode input
AFE_writeRegister(AFE_REG_CH2SET, 0x60); // Normal operation on CH2 and
 gain=12, normal electrode input
AFE_writeRegister(AFE_REG_CH3SET, 0x60); // Normal operation on CH3 and
 gain=12, normal electrode input
```

```
AFE_writeRegister(AFE_REG_CH4SET, 0x60); // Normal operation on CH4 and
 gain=12, normal electrode input
AFE_writeRegister(AFE_REG_CH5SET, 0x60); // Normal operation on CH5 and
 gain=12, normal electrode input
AFE_writeRegister(AFE_REG_CH6SET, 0x60); // Normal operation on CH6 and
 gain=12, normal electrode input
AFE_writeRegister(AFE_REG_CH7SET, 0x60); // Normal operation on CH7 and
 gain=12, normal electrode input
AFE_writeRegister(AFE_REG_CH8SET, 0x60); // Normal operation on CH8 and
 gain=12, normal electrode input

AFE_writeRegister(AFE_REG_RLD_SENSP, 0x00); // RLD : disconnected to the 8
 used inputs (pos) : pour pouvoir avoir 1.65V sur la pin RLD et la prendre
 comme référence des électrodes
AFE_writeRegister(AFE_REG_RLD_SENSN, 0x00); // RLD : disconnected to the 8
 used inputs (neg)

AFE_sendCmd(AFE_CMD_START); // Démarre la conversion

AFE_sendCmd(AFE_CMD_RDATAC); // Enable Read Data Continious Mode

P4DIR |= BIT6; // On init la LED pour voir si la transmission a lieu

__bis_SR_register(GIE); // Enable all Interrupts

//-------------------------------------------------------------------------
// Fin du Setup SPI entre le MSP et l'ADS
//-------------------------------------------------------------------------


//-------------------------------------------------------------------------
// Partie pour l'envoie des données par UART
//-------------------------------------------------------------------------

while(1)
{
    if(flag == 1) // Si les 32 échantillons de chaque canal sont prêts à être
     envoyés
    {
        TX_data_buffer[0]=254; // Initialisation de la variable à transmettre
         avec toutes les valeurs échantillonnées à 1kHz, on envoie un 254 au
         début pour la réception sous Matlab
        TX_data_buffer[449]=0x01; // Envoie un 1 à la fin pour vérifier que le
         paquet de données est correct

        // Pour trier avant matlab: 254, 32 échantillons C1, 32 échantillons
         C2... 32 échantillons C7, 1
        for (j=0; j<nb_echant_canaux; j++)
        {
            // Pour le canal 1
            for (i=1; i<=2; i++)
            {
                TX_data_buffer[i+j*2]=*(ptrlecture+i+j*(7*2));
            }

            // Pour le canal 2
            for (i=3; i<=4; i++)
```

```
                {
                    TX_data_buffer[(nb_echant_canaux*2)+1+
                     (i-3)+j*2]=*(ptrlecture+i+j*(7*2));
                }

                // Pour le canal 3
                for (i=5; i<=6; i++)
                {
                    TX_data_buffer[(nb_echant_canaux*4)+1+
                     (i-5)+j*2]=*(ptrlecture+i+j*(7*2));
                }

                // Pour le canal 4
                for (i=7; i<=8; i++)
                {
                    TX_data_buffer[(nb_echant_canaux*6)+1+
                     (i-7)+j*2]=*(ptrlecture+i+j*(7*2));
                }

                // Pour le canal 5
                for (i=9; i<=10; i++)
                {
                    TX_data_buffer[(nb_echant_canaux*8)+1+
                     (i-9)+j*2]=*(ptrlecture+i+j*(7*2));
                }

                // Pour le canal 6
                for (i=11; i<=12; i++)
                {
                    TX_data_buffer[(nb_echant_canaux*10)+1+
                     (i-11)+j*2]=*(ptrlecture+i+j*(7*2));
                }

                // Pour le canal 7
                for (i=13; i<=14; i++)
                {
                    TX_data_buffer[(nb_echant_canaux*12)+1+
                     (i-13)+j*2]=*(ptrlecture+i+j*(7*2));
                }
            }

            // On envoie
            for(nb = 0; nb < 450; nb++)
            {
                while (!(UCA1IFG&UCTXIFG)); // USCI_A0 TX buffer est t-il prêt ?
                 Lorsque le flag UCTXIFG de UCA1IFG est levé (0) alors on envoie
                UCA1TXBUF = TX_data_buffer[nb]; // Envoie les valeurs par UART à
                 l'ordi
            }
            flag = 0; // Baisse le flag pour les prochains échantillons
        }
    }

    //-------------------------------------------------------------------------
    // Fin de la partie pour l'envoie des données par UART
    //-------------------------------------------------------------------------
}
```

```
// Routine d'interruption utilisée pour récupérer les données SPI à chaque front
 descendant de DRDY
#pragma vector=PORT1_VECTOR // Cette ligne permet de préciser qu'on regarde les
 interruptions sur le port 1
__interrupt void PORT_1(void)
{
    static int n = 0; // Variable propre à la fonction, n'est pas réinitialisée à
     chaque appel de la fonction

    static int index_EMG=0;
    static int index_EMG_2=0;
    static int index_OUT=1;
    static int i=0;

    uint32_t var_EMG=0;

    static int data_EMG[21];
    volatile static int data_EMG_et_etat[24];

    P1IFG &= ~BIT1; // Flag IFG cleared

    toggle ^= BIT0;

    if (toggle == 1)
    {
        AFE_CS_SELECT; // Démarre la communication SPI

        // Doit lire dans le registre de Dout le paquet SPI: 24 bits d'état, 24
         bits canal 1...
        // 24 bits de précision par canal et 24 bits d'états au début: chaque
         buffer est sur 8 bits= 1byte
        for(index_EMG = 0; index_EMG < 24; index_EMG ++)
        {
            UCB1TXBUF = 0xFF; // On écrit dans le buffer de transmission SPI pour
             lancer la clock

            while(UCB1STAT & UCBUSY); // On attend qu'on ait un nouveau byte prêt
             à être lu

            data_EMG_et_etat[index_EMG] = UCB1RXBUF; // On lit dans le buffer qui
             contient Dout (les échantillons des canaux)

            if (index_EMG >= 3) // Dès qu'on passe les 24 bits d'états
            {
                data_EMG[index_EMG_2] = UCB1RXBUF; // On ne prend pas les bytes
                 d'état
                index_EMG_2 ++;
            }
        }

        index_EMG_2=0;

        P1OUT |= BIT2; // Stop la communication SPI

        // On réduit les échantillons à 2 octets plutôt que 3
```

```
        for (i=0; i<7; i++)
        {
            var_EMG = 0;
            volatile uint32_t var1, var2, var3;

            var1 = data_EMG[i*3];
            var2 = data_EMG[(i*3)+1];
            var3 = data_EMG[(i*3)+2];
            var_EMG = (var1 << 16) + (var2 << 8) + (var3);
            var_EMG = (var_EMG >> 8);// --> /256, on réduit à 16 bits (décalage de
             8 bits)

            *(ptrecriture+index_OUT) = (var_EMG & 0xFF00) >> 8; // MSB toujours en
             premier
            *(ptrecriture+index_OUT+1)  = (var_EMG & 0x00FF);
            // Le *(A) sinifie qu'on écrit dans la case qui se trouve à l'adresse
             A.

            index_OUT=index_OUT+2;
        }

        // A ce niveau, ptrecriture (donc buffer 1 ou 2) contient 2 octets/canal
         et donc 1 échantillon/canal pour n=0

        n++; // On incrémente à chaque fois qu'on entre dans l'interruption, c'est
         à dire à chaque fois qu'on prend un échantillon/canal

        // Lorsqu'on arrive à 32 échantillons, on est prêt à envoyer les données
         par UART, donc on lève un flag et on remet n à 0 pour le prochain paquet
         de données
        if(n==32)
        {
            P4OUT ^= BIT6; // Pour faire clignoter la LED et voir si on transmet
            flag = 1;
            n=0;
            index_OUT=1;

            // Partie pour écrire à chaque nouveau n=32 dans un nouveau buffer
            if(double_buffer_id==0)
                {
                    ptrlecture = buffer1;
                    ptrecriture = buffer2;
                    double_buffer_id=1;
                }
            else {
                    ptrlecture = buffer2;
                    ptrecriture = buffer1;
                    double_buffer_id=0;
                }
        }
    }
}
```

Annexe C : Code C++ implémenté sur Raspberry Pi Zero W (Partie 7)

Main.cpp :

```
#include "MySerial.h"
#include "functions.h"
#include "Tke.h"
#include "Filtre_BP_BS.h"
#include "Features.h"
#include "types.h"
#include "classifiercollector.h"

#include <stdlib.h>
#include <stdio.h>
#include <cstdio>
#include <pthread.h>
#include <ctime>
#include <termios.h>

#define MAX_SIZE 960

using namespace std; // Indique quel espace de noms on va utiliser


// IMPORTANTE VARIABLES
const int echant_canaux(64), echant_canaux_reduced(echant_canaux/2), paquet(6),
 nb_canaux(7);
const int taille_paquet((nb_canaux*echant_canaux)+2); //450
const int taille_fenetre(192);
const int number_segment(10);
const int order_ar(4);

const int freq_echant(1000);
float seuil[nb_canaux];

int winning_finger(0);

// MUTEX, CONDITION FOR THREAD
pthread_mutex_t mylock= PTHREAD_MUTEX_INITIALIZER; // Bloque les autres thread
 tant que les tâches du thread actuel ne sont pas finies
pthread_cond_t cond=PTHREAD_COND_INITIALIZER; // Donne le feux vert aux autres
 thread pour voir si ils peuvent faire leurs actions

// STRUCTURE
typedef struct mybuffer_data_t // Structure pour la lecture et l'écriture des data
{
    float **mybuffer;
    int head;
    int tail;
    int data_size; // length of data/channel (32) to load inside buffer (450 by
     450 data)
    int window_size; // length of data/channel (192) at the ouput of buffer
}mybuffer_data_t;
mybuffer_data_t *mybuffer_data;

// POINTERS
unsigned char *data_receive = new unsigned char [taille_paquet]; // Dynamic
 allocation
int *data = new int [taille_paquet];

//INSTANCE
```

```
Features Feat; // On instancie un objet de la classe Features
Filtre_BP_BS Filtre0, Filtre1, Filtre2, Filtre3, Filtre4, Filtre5, Filtre6; // On
 instancie des objets de la classe Filtre
Tke TKE(nb_canaux); // On instancie un objet de la classe Tke
ClassifierCollector mywrapper; // On instancie un objet de la classe
 ClassifierCollector

// KEYBOARD AND TERMINAL
// préparer l'algorithme à recevoir un évenement sur le clavier
struct termios orig_termios;
void reset_terminal_mode();
void set_conio_terminal_mode();
int kbhit();
int getch();
int ch=0;

// USB
string usb_name;

// SIGNAL AND CLOCK
int quit_loop=0;
int make_calibration=0;
unsigned int incr=0;
int flag(0);


void* read_data(void* arg)
{// Lit les données UART et les écrit dans un buffer (circulaire) de taille 960
 (MAX_SIZE)

    // INITIALIZATION OF USEFUL VARIABLES
    const int bit_start(254);
    const int bit_stop(1);
    int next=0;
    // END INITIALIZATION

    // DYNAMICAL ALLOCATION
    float **c=(float**)calloc(nb_canaux, sizeof(float*));
    for (int i=0; i<nb_canaux; i++)
        c[i]=(float*)calloc(taille_fenetre, sizeof(float));
    // END DYNAMICAL ALLOCATION

    // OPEN UART
    MySerial board(usb_name, 921600); // Baud Rate de 921600 bits/s
    cout << "COM port open:" << board.isOpen() << endl;
    // END OPEN UART


    while(quit_loop==0) // Launch Routine
    {
        // KEYBOARD
        // Gestion des évènements claviers
        ch=0;
        set_conio_terminal_mode();
        if(kbhit())
            ch=getch();
```

```
reset_terminal_mode();
// END KEYBOARD

// START READ DATA FROM UART
if (flag==1) // Si l'acquisition a déjà été lancée, on lit 450 données
    board.receive(data_receive, taille_paquet);

else // Premier lancement: on synchronise la lecture du buffer de la Pi
 avec l'envoie des paquets par le MSP
{
    while (flag==0)
    {
        data_receive[0] = 0;

        while(data_receive[0] != bit_start) // On lit 1 donnée/donnée
         jusqu'à ce qu'on trouve le bit de début
        {
            board.receive(data_receive, 1);
        }

        // READ DATA FROM UART -> 450-1 samples
        board.receive(data_receive+1, taille_paquet-1); // On lit le reste
         des données

        if ((int)data_receive[taille_paquet-1] == bit_stop) // Si le bit
         de fin est à la bonne place
            flag=1;
    }
}

if ((int)data_receive[taille_paquet-1] == bit_stop && (int)data_receive[0]
 == bit_start) // Si les bits de début et de fin sont à la bonne place
{
    for (int i=0; i<taille_paquet; i++)
        data[i] = (int)data_receive[i]; // contient aussi les bits de
         début et de fin
    // END READ DATA FROM UART

    // SORT DATA AND RETURN 7 ARRAYS: c1 ... c7 in one array of array c
    sort_canal(data, echant_canaux, nb_canaux, c);
    // END SORT DATA AND RETURN 7 ARRAYS: c1 ... c7

    // FILL SHARED BUFFER:
    pthread_mutex_lock(&mylock); // Lock

    next=(mybuffer_data->head+mybuffer_data->data_size)%MAX_SIZE; // On
     incrémente next

    for (int i=0; i<nb_canaux; i++) // On remplit le buffer mybuffer avec
     les 7 petits buffers triés
    {
        if (next==0) next=MAX_SIZE;
        int k =0;

        for(int j=mybuffer_data->head; j<next; j++)
        {
            mybuffer_data->mybuffer[i][j]=c[i][k];
```

```
                k++;
            }
        }

        mybuffer_data->head=next%MAX_SIZE; // On incrémente la tête

        // BUTTON EXIT: E
        if (ch==E_KEY)
        {
            ch=0;
            quit_loop=1;
        }
        // END BUTTON EXIT

        // CALIBRATION 5s, CALCULATION THRESHOLD FOR OFF-SET DETECTION, BUTTON
         Q
        // 4992 données sont récupérées
        if (ch==Q_KEY && make_calibration==0)
        {
            make_calibration=1;
            ch=0;
            cout<< "Calibration of 5s en cours, don't move of position rest"
             << endl;
            incr=0;
        }

        if(make_calibration==1)
        {
            incr++;
            if(incr==156)
            {
                make_calibration=0;
                ch=0;
                cout << "End of Calibration" << endl;
                for (int i=0; i<nb_canaux; i++)
                    cout << " Seuil of canal " << i << "= " <<
                     (double)seuil[i] << endl;
            }
        }
        // END CALIBRATION 5s

        // On dit à l'autre thread qu'il peut faire un check car on a ajouté
         des nouvelles données
        pthread_cond_signal(&cond);
        pthread_mutex_unlock(&mylock); // Unlock
    }
}

// FREE MEMORY
for (int i=0; i<nb_canaux; i++)
    free(c[i]);

free(c);
// END FREE MEMORY

cout << "thread read exit" << endl;
pthread_exit(NULL);
```

```
}


void* treatment_data(void* arg)
{// Lit (récupère) les 192 données / canal (lorsqu'elles sont disponibles) du
 buffer du dessus et termine les traitements

    // INITIALIZATION OF USEFUL VARIABLES
    int next=0;
    int cond0(0), cond1(0), cond2(0), cond3(0), cond4(0), cond5(0), cond6(0);
    // END INITIALIZATION

    // DYNAMICAL ALLOCATION
    float **output, **output_TKE;
    output=(float**)calloc(nb_canaux, sizeof(float*));
    output_TKE=(float**)calloc(nb_canaux, sizeof(float*));

    for (int i=0; i<nb_canaux; i++)
    {
        output[i]=(float*)calloc(taille_fenetre, sizeof(float));
        output_TKE[i]=(float*)calloc(taille_fenetre, sizeof(float));
    }
    // END DYNAMICAL ALLOCATION


    while(quit_loop==0) // Launch Routine
    {
        // INITIALIZATION OF USEFUL VARIABLES
        cond0=cond1=cond2=cond3=cond4=cond5=cond6=0;
        // END INITIALIZATION

        pthread_mutex_lock(&mylock); // Lock

        next=(mybuffer_data->tail+mybuffer_data->window_size)%MAX_SIZE; // On
         incrémente next

        while(((next+mybuffer_data->data_size) != mybuffer_data->head  &&
         quit_loop==0 )||
                (mybuffer_data->head==mybuffer_data->tail && mybuffer_data-
                 >head==0 && quit_loop==0)) // Tant qu'on n'a pas 192 données /
                 canal de dispo
        {
            pthread_cond_wait(&cond,&mylock); // On attend jusqu'à ce que la tête
             soit à 192 + un paquet de 32 (car tête et queue ne doivent jamais se
             suivre)
        }

        // RECOVER 192 data/channel in output
        for (int i=0; i<nb_canaux; i++)
        {
            int k=0;
            if (next==0) next=MAX_SIZE;

            for(int j=mybuffer_data->tail; j<next; j++)
            {
```

```
        output[i][k]=(mybuffer_data->mybuffer[i][j]*2.4)/32767; // On
         remet les valeurs à l'échelle par rapport à la plage de
         conversion de l'ADS
        k++;
    }
}
// END RECOVER 192 data/channel in output

mybuffer_data->tail=(next%MAX_SIZE); // On incrémente la tail
pthread_mutex_unlock(&mylock); // Unlock


// FILTERING
Filtre0.filtrer(output[0], taille_fenetre);
Filtre1.filtrer(output[1], taille_fenetre);
Filtre2.filtrer(output[2], taille_fenetre);
Filtre3.filtrer(output[3], taille_fenetre);
Filtre4.filtrer(output[4], taille_fenetre);
Filtre5.filtrer(output[5], taille_fenetre);
Filtre6.filtrer(output[6], taille_fenetre);
// END FILTERING

// TKE
TKE.calculation_TKE(output, nb_canaux, taille_fenetre, output_TKE);
//END TKE

if(make_calibration==0)
{
    // ONSET DETECTION
    for (int i=0; i<taille_fenetre; i++)
    {
        if (output_TKE[0][i]<seuil[0])
            cond0=cond0+1;

        if (output_TKE[1][i]<seuil[1])
            cond1=cond1+1;

        if (output_TKE[2][i]<seuil[2])
            cond2=cond2+1;

        if (output_TKE[3][i]<seuil[3])
            cond3=cond3+1;

        if (output_TKE[4][i]<seuil[4])
            cond4=cond4+1;

        if (output_TKE[5][i]<seuil[5])
            cond5=cond5+1;

        if (output_TKE[6][i]<seuil[6])
            cond6=cond6+1;
    }

    if (cond0==taille_fenetre && cond1==taille_fenetre &&
     cond2==taille_fenetre && cond3==taille_fenetre &&
     cond4==taille_fenetre && cond5==taille_fenetre &&
     cond6==taille_fenetre)
```

```
    {
        winning_finger=1; // Neutral movement
        cout << " ********** neutral **********" << endl;
    }
    // END ONSET DETECTION

    else // si on a repéré une activité musculaire
    {
        // FEATURES CALCULATION
        Feat.calculation_features(output, taille_fenetre, nb_canaux);
        // END FEATURES CALCULATION

        // PREDICTION
        // Ordre du vecteur features = ordre de la database
        mywrapper.Clean_vector();
        mywrapper.Fill_vector(Feat.features_vector, 1050);

        winning_finger=mywrapper.Classifier_class();
        // END PREDICTION

        // DISPLAY WINNING FINGER
        if(winning_finger==1)
            cout<<"Open Hand"<<endl;
        else if (winning_finger==2)
            cout<<"Thumb Flexion"<<endl;
        else if (winning_finger==3)
            cout<<"Index Flexion"<<endl;
        else if (winning_finger==4)
            cout<<"Major Flexion"<<endl;
        else if (winning_finger==5)
            cout<<"Annular Flexion"<<endl;
        else if (winning_finger==6)
            cout<<"Auricular Flexion"<<endl;
        else if (winning_finger==7)
            cout<<"Simple Pliers"<<endl;
        else if (winning_finger==8)
            cout<<"Complex Pliers"<<endl;
        else if (winning_finger==9)
            cout<<"Point Index"<<endl;
        else if (winning_finger==10)
            cout<<"3 Finger Flexion"<<endl;
        else if (winning_finger==11)
            cout<<"Closed Hand"<<endl;
        else if (winning_finger==12)
            cout<<"Cylindrical Grip"<<endl;
        else if (winning_finger==13)
            cout<<"Lateral Grip"<<endl;
        else if (winning_finger==0)
            cout<<"Error finger"<<endl;
        // END DISPLAY WINNING FINGER
    }
}

else // si on doit faire la calibration
{
    for(int i=0; i<nb_canaux; i++)
    {
```

```
                for(int j=0; j<taille_fenetre; j++)
                {
                    if(output_TKE[i][j]>=seuil[i])
                        seuil[i]=output_TKE[i][j];
                }
            }
        }
    } // END Routine


    // FREE MEMORY
    for (int i=0; i<nb_canaux; i++)
    {
        free(output[i]);
        free(output_TKE[i]);
    }

    free(output);
    free(output_TKE);
    // END FREE MEMORY

    cout << "thread treatment exit" << endl;
    pthread_exit(NULL);
}


int main()
{
    // INIT USEFUL VARIABLES
    for (int i=0; i<nb_canaux; i++)
        seuil[i]=0.;
    // END INIT VARIABLES

    // DYNAMICAL ALLOCATION
    mybuffer_data=(mybuffer_data_t*)malloc(sizeof(mybuffer_data_t));

    if (mybuffer_data != NULL)
    {
        mybuffer_data->mybuffer=(float**)calloc(nb_canaux, sizeof(float*));

        for (int i=0; i<nb_canaux; i++)
            mybuffer_data->mybuffer[i]=(float*)calloc(MAX_SIZE, sizeof(float));

        if(mybuffer_data->mybuffer != NULL)
        {
            mybuffer_data->head=0;
            mybuffer_data->tail=0;
            mybuffer_data->data_size=32;
            mybuffer_data->window_size=192;
        }

        else
            exit(-1);
    }

    else
        exit(-1);
```

```
// END DYNAMICAL ALLOCATION

// DISPLAY and LOAD DATABASE FOLDER
char* path_p=new char[155];
path_p[0]='\0';

charger_database(path_p);

const char* path=path_p;
mywrapper.Send_path_to_wrapper(path);
cout << "order sent" << endl;

delete [] path_p;
path=0;
// END DISPLAY and LOAD DATABASE FOLDER

// CONFIG USB PORT
int result=0;
cout << "Available Ports:" << endl;
result = system("ls -l /dev/ttyUSB*");

if (result != 0)
{
    cout << "No USB devices were found" << endl;
    //exit(-1);
    usb_name="/dev/ttyUSB0";
}

else
{
    cout << "Enter COM port [/dev/ttyUSB*]:" << endl;
    cin >> usb_name;
}
// END CONFIG USB PORT

cout << GREEN << "Please press d to begin" << RESET << endl;

// WAIT D TO START
set_conio_terminal_mode();
while(ch!=D_KEY)
{
    if(kbhit())
        ch=getch();

    usleep(100000);
}
reset_terminal_mode();
// END WAIT D TO START

// CONFIG AND LAUNCH THREAD
pthread_t thread[2];

pthread_create(&thread[0],NULL, read_data,NULL);
pthread_create(&thread[1],NULL, treatment_data,NULL);

pthread_join(thread[0], NULL);
pthread_join(thread[1], NULL);
```

```
    // END CONFIG AND LAUNCH THREAD

    // A partir de là, les threads sont lancés et on sort du main jusqu'à la
     fermeture du programme
    // On termine le main lorsque la condition d'arrêt dans les threads est vraie

    cout << "thread exit succeed" << endl;

    // FREE MEMORY
    delete[] data_receive;
    data_receive=0;
    delete[] data;
    data=0;

    for(int i=0; i<nb_canaux; i++)
        free(mybuffer_data->mybuffer[i]);

    free(mybuffer_data->mybuffer);
    free(mybuffer_data);

    mywrapper.Close_bridge();
    // END FREE MEMORY

    cout << "Program Success" <<endl;
    return 0;
}


void reset_terminal_mode()
{
    tcsetattr(0,TCSANOW, &orig_termios);
}

void set_conio_terminal_mode()
{
    struct termios new_termios;
    // copy now - later
    tcgetattr(0,&orig_termios);
    memcpy(&new_termios,&orig_termios,sizeof(new_termios));

    // cleanup register
    atexit(reset_terminal_mode);
    cfmakeraw(&new_termios);
    tcsetattr(0,TCSANOW,&new_termios);
}

int kbhit()
{
    struct timeval tv={0L,0L};
    fd_set fds;
    FD_ZERO(&fds);
    FD_SET(0,&fds);
    return select(1,&fds,NULL,NULL,&tv);
}

int getch()
{
```

```
    int r;
    unsigned char c;
    if((r=read(0,&c,sizeof(c)))<0)
        return r;

    else
        return c;
}
```

MySerial.h :

```
#ifndef DEF_MYSERIAL
#define DEF_MYSERIAL

#include <string>
#include <iostream>
#include <unistd.h>


class  MySerial
{
    public:

    MySerial(std::string deviceName, int baud);
    ~MySerial();

    int receive(unsigned char  *data, int len);
    bool isOpen(void);
    void myClose(void);
    bool myOpen(std::string deviceName, int baud);
    bool numberByteRcv(int &bytelen);

    private:

    int handle; // 0: Port open -1: Port close
    std::string  deviceName;
    int baud;

};
#endif
```

MySerial.cpp :

```
#include "MySerial.h"

#include <asm/termbits.h>
#include <sys/ioctl.h>
#include <fcntl.h>


using namespace std;


MySerial::MySerial(string deviceName, int baud)
{
   handle=-1;
   myOpen(deviceName,baud);
}


MySerial::~MySerial()
{
  if(handle >=0)
      myClose();
}


void MySerial::myClose(void)
{// Close USB device
   if(handle >=0)
      close(handle);
   handle = -1;
}


bool MySerial::myOpen(string deviceName, int baud)
{// Open USB device
    struct termios tio; // Use struct in termios (C)
    struct termios2 tio2;
    this->deviceName=deviceName;
    this->baud=baud;

    // Open USB port
    handle  = open(this->deviceName.c_str(),O_RDWR | O_NOCTTY);

    if(handle <0)
       return false;

    // Config port USB with termios
    tio.c_cflag =  B921600 | CS8 | CLOCAL | CREAD;
    tio.c_iflag = IGNPAR ;
    tio.c_oflag = 0;
    tio.c_lflag = 0;
    tio.c_cc[VMIN]=0;
    tio.c_cc[VTIME]=10; // time out every 0.1/0.5 sec
    ioctl(handle,TCSETS,&tio);

    ioctl(handle,TCGETS2,&tio2);
    tio2.c_cflag &= ~CBAUD;
    tio2.c_cflag |= BOTHER;
```

```
    tio2.c_ispeed = baud;
    tio2.c_ospeed = baud;
    ioctl(handle,TCSETS2,&tio2);

    // Flush buffer
    ioctl(handle,TCFLSH,TCIOFLUSH);

    return true;
}


bool MySerial::isOpen(void)
{// Return if USB is open
   return(handle >=0);
}


int  MySerial::receive(unsigned char  *data, int len)
{// UART protocol need char to function
   if(!isOpen()) return -1;

   // this is a blocking receives
   int lenRCV=0;
   while(lenRCV < len)
     {
       int rlen = read(handle,&data[lenRCV],len - lenRCV);
       lenRCV+=rlen;
     }
   return  lenRCV;
}


bool MySerial::numberByteRcv(int &bytelen)
{// Return the number of received bits
   if(!isOpen()) return false;
   ioctl(handle, FIONREAD, &bytelen);
   return true;
}
```

Filtre_BP_BS.h :

```
#ifndef DEF_FILTRE_BP_BS
#define DEF_FILTRE_BP_BS

#include <stdlib.h>
#include <stdio.h>
#include <string>

#define ORDER_FILTER 2


class Filtre_BP_BS
{
    public:

    Filtre_BP_BS();
    ~Filtre_BP_BS();

    void filtrer_BP(float *canal_number, int lenght_canal);
    void filtrer_BS(float *canal_number, int lenght_canal);
    void filtrer(float *canal_number, int lenght_canal);

    private:

    // Filtering routine variables
    float m_y1, m_y2, m_y3, m_y4;
    float m_z1, m_z2, m_z3, m_z4;
    float m_w1, m_w2;
    float m_x1, m_x2;

    // Bandpass Butter IIR
    double m_num_bp[ORDER_FILTER+3], m_den_bp[ORDER_FILTER+3];
    // Notch IIR
    double m_num_bs[ORDER_FILTER+1], m_den_bs[ORDER_FILTER+1];

};

#endif
```

Filtre_BP_BS.cpp :

```
#include "Filtre_BP_BS.h"

using namespace std;


Filtre_BP_BS::Filtre_BP_BS()
{
    m_y1=0;
    m_y2=0;
    m_y3=0;
    m_y4=0;

    m_z1=0;
    m_z2=0;
    m_z3=0;
    m_z4=0;

    m_w1=0;
    m_w2=0;

    m_x1=0;
    m_x2=0;

    // Set up filtering
    m_num_bp[0]=0.875183092438135;
    m_num_bp[1]=0;
    m_num_bp[2]=-1.75036618487627;
    m_num_bp[3]=0;
    m_num_bp[4]=0.875183092438135;

    m_den_bp[0]=1;
    m_den_bp[1]=0.0885412209797640;
    m_den_bp[2]=-1.73124095818815;
    m_den_bp[3]=-0.0676598631639336;
    m_den_bp[4]=0.766006600943264;

    m_num_bs[0]=0.842646292224708;
    m_num_bs[1]=-1.557390190740875;
    m_num_bs[2]=0.832369385605689;

    m_den_bs[0]=1;
    m_den_bs[1]=-1.557390190740875;
    m_den_bs[2]=0.675015677830398;
}


Filtre_BP_BS::~Filtre_BP_BS()
{}


void Filtre_BP_BS::filtrer_BP(float *canal_number, int lenght_canal)
{
    float y0(0);
    float z0(0);

    for (int i=0; i<lenght_canal; i++)
    {
```

```
        // Band Pass
        y0=canal_number[i];
        z0 = m_num_bp[0]*y0 + m_num_bp[1]*m_y1 + m_num_bp[2]*m_y2 +
            m_num_bp[3]*m_y3 + m_num_bp[4]*m_y4 - (m_den_bp[1]*m_z1 +
            m_den_bp[2]*m_z2 + m_den_bp[3]*m_z3 + m_den_bp[4]*m_z4);

        m_y4 = m_y3;
        m_y3 = m_y2;
        m_y2 = m_y1;
        m_y1 = y0;

        m_z4 = m_z3;
        m_z3 = m_z2;
        m_z2 = m_z1;
        m_z1 = z0;

        canal_number[i]=z0;
    }
}


void Filtre_BP_BS::filtrer_BS(float *canal_number, int lenght_canal)
{
    float x0(0);
    float w0(0);

    for (int i=0; i<lenght_canal; i++)
    {
        // Band Stop (Notch)
        x0=canal_number[i];
        w0 = m_num_bs[0]*x0 + m_num_bs[1]*m_x1 + m_num_bs[2]*m_x2 -
            (m_den_bs[1]*m_w1 + m_den_bs[2]*m_w2);

        m_x2 = m_x1;
        m_x1 = x0;

        m_w2 = m_w1;
        m_w1 = w0;

        canal_number[i]=w0;
    }
}


void Filtre_BP_BS::filtrer(float *canal_number, int lenght_canal)
{
    float y0(0);
    float z0(0);

    float x0(0);
    float w0(0);

    for (int i=0; i<lenght_canal; i++)
    {
        // Band Pass
        y0=canal_number[i];
        z0 = m_num_bp[0]*y0 + m_num_bp[1]*m_y1 + m_num_bp[2]*m_y2 +
```

```
            m_num_bp[3]*m_y3 + m_num_bp[4]*m_y4 - (m_den_bp[1]*m_z1 +
            m_den_bp[2]*m_z2 + m_den_bp[3]*m_z3 + m_den_bp[4]*m_z4);

        m_y4 = m_y3;
        m_y3 = m_y2;
        m_y2 = m_y1;
        m_y1 = y0;

        m_z4 = m_z3;
        m_z3 = m_z2;
        m_z2 = m_z1;
        m_z1 = z0;

        // Band Stop (Notch)
        x0=z0;
        w0 = m_num_bs[0]*x0 + m_num_bs[1]*m_x1 + m_num_bs[2]*m_x2 -
            (m_den_bs[1]*m_w1 + m_den_bs[2]*m_w2);

        m_x2 = m_x1;
        m_x1 = x0;

        m_w2 = m_w1;
        m_w1 = w0;

        canal_number[i]=w0;
    }
}
```

Tke.h :

```
#ifndef DEF_TKE
#define DEF_TKE

#include <stdlib.h>
#include <stdio.h>
#include <cmath>


class Tke
{
    public:

    Tke(int nb_canaux);
    ~Tke();

    void calculation_TKE(float **canal_number,int nb_canaux, int lenght_canal,
     float **canal_number_TKE);

    private:

    // TKE routine variable
    float *m_xnm;

};

#endif
```

Tke.cpp :

```
#include "Tke.h"

using namespace std;


Tke::Tke(int nb_canaux)
{
    m_xnm=new float[nb_canaux];

    for (int i=0; i<nb_canaux; i++)
        m_xnm[i]=0;
}


Tke::~Tke()
{
    delete [] m_xnm;
    m_xnm=0;
}


void Tke::calculation_TKE(float **canal_number,int nb_canaux, int lenght_canal,
 float **canal_number_TKE)
{
    for(int m=0; m<nb_canaux; m++)
    {
        for (int i=0; i<lenght_canal-1; i++)
        {
            canal_number_TKE[m][i] = abs((pow(canal_number[m][i],2))-
             (canal_number[m][i+1]*m_xnm[m]));
            m_xnm[m] = canal_number[m][i];
        }

        canal_number_TKE[m][lenght_canal-1]=canal_number_TKE[m][lenght_canal-2];
    }
}
```

functions.h :

```
#include <stdlib.h>
#include <stdio.h>
#include <cmath>
#include <complex.h>
#include <dirent.h>
#include <iostream>
#include <string>
#include <cstring>
#include <fstream>
#include <errno.h>

#include "types.h"

#define NUMBER_OBSERVATION 10140
#define NUMBER_FEATURES 1050


float complement_2(int var);
void sort_canal(int* array,const int echant_canaux,const int nb_canaux,float** c);
int charger_database(char *path);
```

functions.cpp :

```
#include "functions.h"

using namespace std;


float complement_2(int var)
{
    float var1, var2;
    int mask = 32767; //mask 0x7FFF
    var1 = var & mask;
    var2 = (var >> 15)*(-32768);
    return (var1 + var2);
}


void sort_canal(int* array, const int echant_canaux, const int nb_canaux, float**
 c)
{
    // DYNAMICAL ALLOCATION
    int **c_bis=(int**)calloc(nb_canaux, sizeof(int*));

    for (int i=0; i<nb_canaux; i++)
        c_bis[i]=(int*)calloc(echant_canaux, sizeof(int));

    for (int j=0; j<nb_canaux; j++)
    {
        // Triage des canaux dans l'ordre en enlevant le bit de début et de fin
        for (int i=1; i<echant_canaux+1; i++)
            c_bis[j][i-1]=array[i+(echant_canaux*(j))];

        // Recomposer les 2 morceaux d'échantillons en un
        // On décale vers la gauche de 8 bits le premier échantillon
        for (int b=0; b<echant_canaux/2; b++)
            c[j][b]=complement_2((c_bis[j][b*2]<<8)+c_bis[j][b*2+1]);
    }

    // FREE MEMORY
    for(int i=0; i<nb_canaux; i++)
        free(c_bis[i]);

    free(c_bis);
}


int charger_database(char *path)
{
    DIR* rep = opendir("."); // Current folder
    DIR* database = NULL;

    string database_folder;
    string features_file;
    string classes_file;

    char* path_database = new char [50];

    struct dirent* dossierLu = NULL;
    struct dirent* fichierLu = NULL;
```

```
// OPEN CURRENT FOLDER
if (rep == NULL) /* Si le dossier n'a pas pu être ouvert */
{
    cout<< BOLDRED <<"Fail opening current folder"<<RESET<< endl;
    exit(1);
}

else {
    cout<< BOLDGREEN <<"Current folder was opened"<<RESET <<endl;
}
// END OPEN CURRENT FOLDER


// DISPLAY ALL FOLDER IN CURRENT FOLDER
cout<< BOLDBLUE <<"Display current folder list: "<<RESET <<endl;

while ((dossierLu = readdir(rep)) != NULL)
{
    if (dossierLu->d_type== DT_DIR && (strcmp(dossierLu->d_name,".")
    && strcmp(dossierLu->d_name,"..") &&
    strcmp(dossierLu->d_name,"bin") &&
    strcmp(dossierLu->d_name,".git") &&
    strcmp(dossierLu->d_name,"CMakeFiles")))
        printf("%s\n ", dossierLu->d_name);
}
// END DISPLAY ALL FOLDER IN CURRENT FOLDER

cout << endl;

// SELECT and OPEN DATABASE
cout<<BOLDMAGENTA<<"Enter the database folder selected"<<RESET<<endl;
cin>>database_folder;

strcat(path,database_folder.c_str());
strcat(path,",");

path_database[0]='\0';
strcat(path_database,"./");
strcat(path_database,database_folder.c_str());

database=opendir(path_database);

if (database == NULL) /* Si le dossier n'a pas pu être ouvert */
{
    cout<<BOLDRED<<"Fail opening database folder"<< RESET<<endl;
    perror("");
    exit(1);
}
else
    cout<< BOLDGREEN <<"Database folder was opened"<<RESET<<endl;
// END SELECT and OPEN DATABASE


// DISPLAY ALL FILES IN DATABASE FOLDER
cout<<BOLDBLUE<<"Display file list:"<<RESET<<endl;
while ((fichierLu = readdir(database)) != NULL)
```

```
    {
        if ((strcmp(fichierLu->d_name,".") &&
       strcmp(fichierLu->d_name,"..") && strcmp(fichierLu->d_name,"bin")
       && strcmp(fichierLu->d_name,".git") &&
       strcmp(fichierLu->d_name,"CMakeFiles")))
            printf("'%s'\n", fichierLu->d_name);
    }
    // END DISPLAY ALL FILES IN DATABASE FOLDER


    // SELECT and OPEN FILES IN DATABASE
    // Matlab va enregistrer la matrice Features et le vecteur Classes
    // dans 2 fichiers.txt séparés. Nous n'aurons plus qu'à lire
    // ces deux fichiers dans le répertoire de la database
    cout<<BOLDMAGENTA<<"Enter the file selected for Features"<<RESET<<endl;
    cin>>features_file;

    strcat(path,features_file.c_str());
    strcat(path,",");

    cout<<BOLDMAGENTA<<"Enter the file selected for Classes"<<RESET<<endl;
    cin>>classes_file;

    strcat(path,classes_file.c_str());

    cout<<BOLDGREEN<<"Database folder was closed"<<RESET<<endl;


    // FREE MEMORY
    delete[] path_database;
    path_database=0;
    // END FREE MEMORY

    return 0;
}
```

Features.h :

```
#ifndef DEF_FEATURES
#define DEF_FEATURES

#include <stdlib.h>
#include <stdio.h>
#include <cmath>
#include <complex.h>
#include <fftw3.h>
#include <iostream>

#include <algorithm>

#define REAL 0
#define IMAG 1
#define ORDER_AR 4
#define LENGTH(x)  (sizeof(x)/ sizeof((x)[0]))


class Features
{
    public:

        Features();
        ~Features();

        int sign(float x);
        double mean(float* array, int length);
        double devia(float* array,double mean_o,int length);
        void diff_approx(float* array,int length,float* diff_and_approx);
        double variance(float* array, int length);

        int linsolve(int order,double* pfMatr,double* pfVect,double* pfSolution);

        double mav(float* array,int length);
        int zero_crossing(float* array, int length, float threshold);
        int slope_sign_change(float* array,int length,float threshold);
        double waveform_length(float* array, int length);
        int wilson_amplitude(float* array, int length, float threshold);
        void ar_yule_walker(float* array,int length_array,int order,double*
         autoregressive_coefficient);
        double skewness_value(float* array, int length);
        double iemg(float* array, int length);
        double hjorth_activity(float* array, int length);
        double hjorth_mobility(float* array, int length);
        double hjorth_complexity(float* array, int length);
        double mean_frequency(float* array, int lenght);

        void calculation_features(float **canal_number,int lenght_canal,int
         nb_canaux);

        double *features_vector;

    private:

        int m_size_window;
        int m_size_segment;
```

```
        int m_size_shift_segment;
        int m_number_segment;

        float m_threshold_feature;
        int m_cut_frequency;

        double c_mav, c_h_mobility,c_h_complexity,c_h_activity,
            c_waveform_l,c_mean_freq,c_skewness,c_iemg;
        int c_zc, c_ssc,c_wamp;
        double *c_ar_coeff;

        float *copie_canal;

};

#endif
```

Features.cpp :

```
#include "Features.h"

using namespace std;


Features::Features()
{
    m_size_window=192;
    m_size_segment=48;

    m_size_shift_segment=16;
    m_number_segment=(m_size_window/m_size_shift_segment)-2;

    m_threshold_feature=0.0005;
    m_cut_frequency=500;

    copie_canal = new float [m_size_window];

    c_ar_coeff=(double*)calloc(ORDER_AR,sizeof(double));

    features_vector=(double*)calloc(1050,sizeof(double));
}


Features::~Features()
{
    delete[] copie_canal;
    copie_canal=0;

    free(c_ar_coeff);

    free(features_vector);
}



int Features::sign(float x)
{
    int y;
    y=(x>=0) + (-1)*(x<0);

    return y;
}


double Features::mean(float* array, int length)
{
    double mean_o=0.;

    for (int i=0; i< length; i++)
        mean_o+=array[i];

    return mean_o/(double)length;
}


double Features::devia(float* array,double mean_o, int length)
```

```
{
    double stdandard_deviation=0.;

    for (int i=0; i<length; i++)
        stdandard_deviation+= pow(array[i] - mean_o, 2);

    return sqrt(stdandard_deviation/(double)length);
}


void Features::diff_approx(float* array,int length,float* diff_and_approx)
{
    for (int i=0; i<length-1; i++)
        diff_and_approx[i]=array[i+1]-array[i];
}


double Features::variance(float* array, int length)
{
    double variance_return=0.;
    double mean_o=mean(array, length);

    for (int i=0; i<length; i++)
        variance_return+=pow(abs(array[i]-mean_o),2);

    return variance_return/((double)(length-1));
}


int Features::linsolve(int nDim/*order*/, double* pfMatr, double* pfVect, double*
 pfSolution)
{
  double fMaxElem;
  double fAcc;

  int i, j, k, m;

  for(k=0; k<(nDim-1); k++) // base row of matrix
  {
    // search of line with max element
    fMaxElem = fabs( pfMatr[k*nDim + k] );
    m = k;

    for(i=k+1; i<nDim; i++)
    {
      if(fMaxElem < fabs(pfMatr[i*nDim + k]) )
      {
        fMaxElem = pfMatr[i*nDim + k];
        m = i;
      }
    }

    // permutation of base line (index k) and max element line(index m)
    if(m != k)
    {
      for(i=k; i<nDim; i++)
      {
```

```
        fAcc               = pfMatr[k*nDim + i];
        pfMatr[k*nDim + i] = pfMatr[m*nDim + i];
        pfMatr[m*nDim + i] = fAcc;
      }

      fAcc = pfVect[k];
      pfVect[k] = pfVect[m];
      pfVect[m] = fAcc;
    }

    if( pfMatr[k*nDim + k] == 0.) return 1;

    // triangulation of matrix with coefficients
    for(j=(k+1); j<nDim; j++) // current row of matrix
    {
      fAcc = - pfMatr[j*nDim + k] / pfMatr[k*nDim + k];

      for(i=k; i<nDim; i++)
        pfMatr[j*nDim + i] = pfMatr[j*nDim+i] + fAcc*pfMatr[k*nDim+i];

      pfVect[j] = pfVect[j]+fAcc*pfVect[k]; //free member recalculation
    }
  }

  for(k=(nDim-1); k>=0; k--)
  {
    pfSolution[k] = pfVect[k];

    for(i=(k+1); i<nDim; i++)
      pfSolution[k] -= (pfMatr[k*nDim + i]*pfSolution[i]);

    pfSolution[k] = pfSolution[k] / pfMatr[k*nDim + k];
  }

  return 0;
}



double Features::mav(float* array, int length)
{
    double iemg_value=0.;
    double mav_value=0.;

    for(int i=0; i<length; i++)
        iemg_value=iemg_value+abs(array[i]);

    if (iemg_value ==0 || length ==0)
        mav_value=0;

    else
        mav_value=iemg_value/length;

    return mav_value;
}
```

```
int Features::zero_crossing(float* array, int length , float threshold)
{
    int number_of_zero_crossing=0;
    int current_sign=sign(array[0]);

    for (int i=1; i<length; i++)
    {
        if(current_sign!=sign(array[i]))
        {
            if (abs(array[i])>threshold)
            {
                number_of_zero_crossing++;
                current_sign=sign(array[i]);
            }
        }
    }

    return number_of_zero_crossing;
}


int Features::slope_sign_change(float* array,int length,float threshold)
{
    int number_of_slope_sign_change=0;
    float slope_sign;

    for(int i=1; i<length-1; i++)
    {
        slope_sign=(array[i]-array[i-1])*(array[i]-array[i+1]);

        if (slope_sign>=threshold)
            number_of_slope_sign_change++;
    }
    return number_of_slope_sign_change;
}


double Features::waveform_length(float* array, int length)
{
    double waveform_length_value=0.0;

    for(int i = 0; i< length-1; i++)
        waveform_length_value+= abs(array[i+1]-array[i]);

    return waveform_length_value;
}


int Features::wilson_amplitude(float* array,int length,float threshold)
{
    int wilson_amplitude_value=0;
    float segment_amplitude=0.;

    for(int i=0; i<length-1; i++)
    {
        if((array[i]-array[i+1])<0)
            segment_amplitude=-(array[i]-array[i+1]);
```

```
        else
            segment_amplitude=-(array[i]-array[i+1]);

        if (segment_amplitude >= threshold)
            wilson_amplitude_value++;
    }
    return wilson_amplitude_value;
}


void Features::ar_yule_walker(float* array, int length_array, int order, double*
 autoregressive_coefficient)
{
    // define order
    int order_ar=order+1;
    // allocate memory for coefficient_to_be_calculated
    double coefficient_to_be_calculated[order_ar+1];
    double toeplitz_matrix[order*order];


    double Transpose_coeff[order_ar-1];
    // for loop to evaluate coefficient_to_be_calculated[0]
    double prod_coefficient_to_be_calculated=0;

    for(int i=0; i<length_array; i++)
        prod_coefficient_to_be_calculated+=(array[i]*array[i]);

    if (length_array==0 || prod_coefficient_to_be_calculated==0)
        coefficient_to_be_calculated[0]=0;

    else
        coefficient_to_be_calculated[0]=prod_coefficient_to_be_calculated/
         length_array;


    // next loop for index from 2 to order_ar+1
    for(int i=1; i<order_ar+1; i++)
    {
        // allocate factor_a for each loop
        double factor_a[length_array-i];

        for(int j=0; j<(int)LENGTH(factor_a); j++)
            factor_a[j]=array[j];

        // allocate factor_a for each loop
        double factor_b[length_array-i];
        int counter=0;

        for(int k=i; k<(int)LENGTH(factor_b)+i; k++)
        {
            factor_b[counter]=array[k];
            counter++;
        }

        // initialize prod_coefficient_to_be_calculated in order to make product
         term by term
```

```
        prod_coefficient_to_be_calculated=0;

        for(int j=0; j<length_array-i; j++)
            prod_coefficient_to_be_calculated+=factor_a[j]*factor_b[j];

        coefficient_to_be_calculated[i]=prod_coefficient_to_be_calculated/
         length_array;
    }

    // toeplitz matrix
    for(int i=0; i<order; i++)
    {
        for(int j=0;  j< order; j++)
        {
            // create symetric matrix -> second part isn't conjugate of vector ->
             real vector
            if (i-j>=0)
                toeplitz_matrix[i*order+j]=coefficient_to_be_calculated[i-j];

            if (j-i>0)
                toeplitz_matrix[i*order+j]=coefficient_to_be_calculated[j-i];
        }
    }

    //return autoregressive_coefficient;
    for (int i=1; i<order_ar; i++)
        Transpose_coeff[i-1]=coefficient_to_be_calculated[i];

    int res;

     res=linsolve(order,toeplitz_matrix,Transpose_coeff ,autoregressive_coefficien
     t);

    if (res)
    {
        printf("Error\n");
        exit(-1);
    }
}


double Features::skewness_value(float* array, int length)
{
    double mean_o=0.;
    double standard_deviation=0.;
    double skewness_value_to_return=0.;
    // mean and std
    mean_o=mean(array,length);
    standard_deviation=devia(array,mean_o,length);

    for (int i=0; i< length; i++)
    {
        if(standard_deviation!=0)
            skewness_value_to_return+=pow(((array[i]-mean_o)/standard_deviation),
             3);
    }
```

```
    if(length==0 || skewness_value_to_return==0)
        skewness_value_to_return=0;

    else
        skewness_value_to_return=skewness_value_to_return/length;

    return skewness_value_to_return;
}


double Features::iemg(float* array, int length)
{
    double iemg_value=0.0;

    for(int i=0; i<length; i++)
        iemg_value=iemg_value+abs(array[i]);

    return iemg_value;
}


double Features::hjorth_activity(float* array, int length)
{
        return variance(array,length);
}


double Features::hjorth_mobility(float* array, int length)
{
    // diff matlab function
    float diff_return[length-1];
    diff_approx(array,length,diff_return);

    // var
    double diff_variance=variance(diff_return,length-1);

    // hjort_activity_value
    double hjort_activity_value=variance(array,length);

    // return of the function
    if(diff_variance<=0 || hjort_activity_value <=0)
        return 0;

    else
        return sqrt(diff_variance/hjort_activity_value);
}


double Features::hjorth_complexity(float* array, int length)
{
    // diff
    float diff_return[length-1];
    diff_approx(array,length,diff_return);

    // diff mobility
    double diff_mobility=hjorth_mobility(diff_return,length-1);
    double mobility=hjorth_mobility(array,length);
```

```
    if(diff_mobility==0 || mobility==0)
        return 0;

    else
        return diff_mobility/mobility;
}


double Features::mean_frequency(float* array, int length)
{
    double transform[length];
    double transform_useful[length/2];
    double power_spectral_density[length/2];
    double frequency[length/2];

    double sum_spectrum=0.;
    double sum_product=0.;

    // To do fftshift(fft(array,lenght),lenght)
    fftw_complex *x;
    fftw_complex *y;

    // Input
    x = (fftw_complex *)fftw_malloc(sizeof(fftw_complex) * length);
    // Output
    y = (fftw_complex *)fftw_malloc(sizeof(fftw_complex) * length);

    for (int j=0; j<length; j++)
    {
       x[j][REAL] = (double)array[j];
       x[j][IMAG] = 0;
    }

    fftw_plan plan = fftw_plan_dft_1d(length, x, y, FFTW_FORWARD, FFTW_ESTIMATE);
    fftw_execute(plan);
    fftw_destroy_plan(plan);
    fftw_cleanup();

    /* FFT shift for complex data*/
    if (length%2==0)
        rotate(y[0],y[length >>1],y[length]);

    else
        rotate(y[0],y[(length >>1)+1],y[length]);

    for (int m=0; m<length; m++)
        transform[m] = (double) (y[m][REAL]);


    fftw_free(x);
    fftw_free(y);

    for (int i=0; i<(length/2); i++)
    {
        transform_useful[i]=transform[i+(length/2)];
        power_spectral_density[i]=pow(abs(transform_useful[i]),2)/m_cut_frequency;
```

```
        frequency[i]=(m_cut_frequency/(length/2-1))*i;
    }

    for (int k=0; k<length/2; k++)
    {
        sum_spectrum = sum_spectrum + power_spectral_density[k];
        sum_product = sum_product + (power_spectral_density[k]*frequency[k]);
    }

    return sum_product/sum_spectrum;
}



void Features::calculation_features(float **canal_number,int lenght_canal, int
 nb_canaux)
{
    for(int l=0; l<nb_canaux; l++)
    {
        for (int j=0; j<m_number_segment; j++)
        {
            int k=0;
            for (int i=m_size_shift_segment*j; i<((m_size_window-1)-
             (m_size_shift_segment*(m_number_segment-(j+1)))); i++)
            {
                copie_canal[k]=canal_number[l][i];
                k++;
            }

            c_mav=mav(copie_canal,m_size_segment);
            c_zc=zero_crossing(copie_canal,m_size_segment,m_threshold_feature);

             c_ssc=slope_sign_change(copie_canal,m_size_segment,m_threshold_featur
             e);
            c_waveform_l=waveform_length(copie_canal,m_size_segment);

             c_wamp=wilson_amplitude(copie_canal,m_size_segment,m_threshold_featur
             e);
            ar_yule_walker(copie_canal,m_size_segment,ORDER_AR,c_ar_coeff);
            c_skewness=skewness_value(copie_canal,m_size_segment);
            c_iemg=iemg(copie_canal,m_size_segment);
            c_h_activity=hjorth_activity(copie_canal,m_size_segment);
            c_h_mobility=hjorth_mobility(copie_canal,m_size_segment);
            c_h_complexity=hjorth_complexity(copie_canal,m_size_segment);
            c_mean_freq=mean_frequency(copie_canal,m_size_segment);


            features_vector[l*(15*m_number_segment)+j]=(double)c_mav;
            features_vector[l*(15*m_number_segment)
             +j+m_number_segment]=(double)c_zc;
            features_vector[l*(15*m_number_segment)
             +j+2*m_number_segment]=(double)c_ssc;
            features_vector[l*(15*m_number_segment)
             +j+3*m_number_segment]=(double)c_waveform_l;
            features_vector[l*(15*m_number_segment)
             +j+4*m_number_segment]=(double)c_wamp;
```

```
            for(int m=0;m<ORDER_AR;m++)
                features_vector[l*(15*m_number_segment)
                 +j*ORDER_AR+5*m_number_segment+m]=(double)c_ar_coeff[m];

            features_vector[l*(15*m_number_segment)
             +j+9*m_number_segment]=(double)c_mean_freq;
            features_vector[l*(15*m_number_segment)
             +j+10*m_number_segment]=(double)c_skewness;
            features_vector[l*(15*m_number_segment)
             +j+11*m_number_segment]=(double)c_iemg;
            features_vector[l*(15*m_number_segment)
             +j+12*m_number_segment]=(double)c_h_activity;
            features_vector[l*(15*m_number_segment)
             +j+13*m_number_segment]=(double)c_h_mobility;
            features_vector[l*(15*m_number_segment)
             +j+14*m_number_segment]=(double)c_h_complexity;
        }
    }
}
```

PythonBridge.h :

```
#include <python2.7/Python.h>
#include <stdlib.h>
#include <stdio.h>
#include <string>
#include <iostream>


class PythonBridge
{
    public :

    PythonBridge();
    ~PythonBridge();

    PyObject *pNameSVM,*pModuleSVM,*pDictSVM,*pClassSVM,*pInstanceSVM;

    int predict(PyObject* vector);
    int load_database_from_Python(PyObject* path);
    int Close_bridge();
};
```

PythonBridge.cpp :

```
#include "PythonBridge.h"

using namespace std;

/* The code below is from Ulysse Project */


PythonBridge::PythonBridge()
{
    pNameSVM=NULL;
    pModuleSVM=NULL;
    pDictSVM=NULL;
    pClassSVM=NULL;
    pInstanceSVM=NULL;
    const char *python_information[] = { "","LDA","LinearDiscriminantA" };

    //Initialize the Python Interpreter
    if (!Py_IsInitialized())
        Py_Initialize();

    //Build the name object
    PyRun_SimpleString("import sys; sys.path.insert(0,'/home/pi/Projet_EMG/')");
    pNameSVM = PyString_FromString(python_information[1]);

    if (pNameSVM==NULL){
        PyErr_Print();
        exit(EXIT_FAILURE);
    }

    //Load the module object
    pModuleSVM = PyImport_Import(pNameSVM);

    if (pModuleSVM==NULL){
        PyErr_Print();
        exit(EXIT_FAILURE);
    }

    //pDict is a borrowed reference
    pDictSVM = PyModule_GetDict(pModuleSVM);

    if (pDictSVM==NULL){
        PyErr_Print();
        exit(EXIT_FAILURE);
    }

    //Build the name of the callable class
    pClassSVM = PyDict_GetItemString(pDictSVM, python_information[2]);

    pInstanceSVM= NULL;

    //Create an instance of the class
    if (PyCallable_Check(pClassSVM))
        pInstanceSVM = PyObject_CallObject(pClassSVM, NULL);

    if (pInstanceSVM==NULL){
        PyErr_Print();
        exit(EXIT_FAILURE);
```

```
    }
}


PythonBridge::~PythonBridge()
{}


int PythonBridge::predict(PyObject* vector)
{
    int classification = 0;
    PyObject *pValue;
    pValue = PyObject_CallMethod(pInstanceSVM,(char*)"_predict",(char*)"O",
     vector);

    if (pValue != NULL)
    {
        classification = PyInt_AsLong(pValue);
        Py_DECREF(pValue);
        Py_DECREF(vector);
    }

    else
    {
        exit(1);
    }

    return classification;
}


int PythonBridge::load_database_from_Python(PyObject* path)
{
    int loaded = 0;
    PyObject *pValue;
    pValue = PyObject_CallMethod(pInstanceSVM,(char*)"_load_database",(char*)"O",
     path);

    if (pValue != NULL)
    {
        loaded = PyInt_AsLong(pValue);
        Py_DECREF(pValue);
    }

    else
    {
        PyErr_Print();
        exit(1);
    }

    return loaded;
}


int PythonBridge::Close_bridge()
{
    Py_DECREF(pInstanceSVM);
```

```
    Py_DECREF(pClassSVM);
    Py_DECREF(pDictSVM);
    Py_DECREF(pModuleSVM);
    Py_DECREF(pNameSVM);
    Py_Finalize();

    return EXIT_SUCCESS;
}
```

classifiercollector.h :

```
#ifndef __CLASSIFIER_COLLECTOR_H_
#define __CLASSIFIER_COLLECTOR_H_

#include "PythonBridge.h"
#include <iostream>
#include <string>

#include <vector>


class ClassifierCollector
{
        public :

            ClassifierCollector();
            ~ClassifierCollector();

            PythonBridge myBridge;
            PyObject *vector;

            int Close_bridge(void);
            int Classifier_class(void);
            int Send_path_to_wrapper(const char *path);
            int Fill_vector(double* data, int lenght);
            int Clean_vector(void);
};

#endif //__CLASSIFIER_COLLECTOR_H_
```

classifiercollector.cpp :

```
#include "classifiercollector.h"

using namespace std;


ClassifierCollector::ClassifierCollector()
{
    vector = PyList_New(0);
}


ClassifierCollector::~ClassifierCollector()
{}


int ClassifierCollector::Close_bridge()
{
    return myBridge.Close_bridge();
}


int ClassifierCollector::Classifier_class(void)
{
    int classification=-1;
    classification = myBridge.predict(vector);

    return classification;
}


int ClassifierCollector::Send_path_to_wrapper(const char *path)
{
    int ex=EXIT_FAILURE;
    PyObject *mypath=PyUnicode_FromString(path);
    ex=myBridge.load_database_from_Python(mypath);
    Py_DECREF(mypath);
    return ex;
}


int ClassifierCollector::Fill_vector(double* data, int length)
{
    PyObject *mylist = PyList_New(length);
    int i;

    for (i = 0; i<length; ++i)
        PyList_SET_ITEM(mylist, i, PyFloat_FromDouble(data[i]));

    vector = Py_BuildValue("(O)", mylist);
    Py_DECREF(mylist);

    return EXIT_SUCCESS;
}


int ClassifierCollector::Clean_vector()
{
```

```
    vector=PyList_New(0);
    return EXIT_SUCCESS;
}
```

types.h :

```
#ifndef __TYPES_H__
#define __TYPES_H__

#define Q_KEY 113
#define D_KEY 100
#define E_KEY 101
#define A_KEY 97

#define RESET "\033[0m"
#define GREEN "\033[32m"
#define RED "\033[31m"
#define BLUE "\033[34m"
#define MAGENTA "\033[35m"
#define BOLDGREEN "\033[1m\033[32m"
#define BOLDRED "\033[1m\033[31m"
#define BOLDBLUE "\033[1m\033[34m"
#define BOLDMAGENTA "\033[1m\033[35m"
#endif //__TYPES_H__
```

LDA.py :

```
from __future__ import division
import numpy as np
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.preprocessing import StandardScaler
from sklearn.externals import joblib
import os


class LinearDiscriminantA (object):
    def __init__(self):
        pass

    def _train_lda(self):
        try :
            self.__lda=joblib.load("%s"%self.__model_name)
            print "Model loaded - %s"%self.__model_name

        except :
            print "Model will be trained, please be patient"

             self.__lda=LinearDiscriminantAnalysis(solver="eigen",shrinkage="auto"
             ,n_components=12) #eigenvalue
            self.__lda.fit(self.__matrix, self.__label)
            joblib.dump(self.__lda,"%s.pkl"%self.__model_name)
            print "Model saved - %s"%self.__model_name
        return 0

    def _predict(self,vector):
        myvector=np.array(vector).reshape(1,-1)
        R=-1
        R=self.__lda.predict(myvector)

        return R

    def _load_database(self,path):
        paths=path.split(',')
        self.__path=paths[0]
        incr=0
        for file in os.listdir("%s/"%self.__path):
            if(file.endswith(".pkl")):
                self.__model_name=os.path.join("%s"%self.__path,file)
                incr+=1
                break

            if(incr==0 or incr >1):
                self.__matrix=np.loadtxt("%s/%s"%(self.__path,paths[1]))
                self.__label=np.loadtxt("%s/%s"%(self.__path,paths[2]))
                self.__model_name="%s/LDA"%self.__path

        self._train_lda()
        return 0

if __name__=='__main__':
    lda=LinearDiscriminantA()
    lda._load_database("22_03_2018_Roxane,matrix_features.txt,vector_classes.txt")
```

CMakeLists.txt :

```
#specify the minimum version of CMake
cmake_minimum_required(VERSION 3.7)

#project's name
project(Project_EMG)

#set the output folder where the program will be created
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -std=c++11 -lpthread -
lpython2.7")
set(CMAKE_BINARY_DIR ${CMAKE_SOURCE_DIR}/bin)
set(EXECUTABLE_OUTPUT_PATH ${CMAKE_BINARY_DIR})
set(LIBRARY_OUTPUT_PATH ${CMAKE_BINARY_DIR}/lib)

# folder which need to be included
include_directories("${PROJECT_SOURCE_DIR}")

# add executable
add_executable(Project_EMG ${PROJECT_SOURCE_DIR}/main.cpp
Features.cpp
Filtre_BP_BS.cpp
functions.cpp
MySerial.cpp
Tke.cpp
PythonBridge.cpp
classifiercollector.cpp
)

#link libraries
target_link_libraries(Project_EMG -lpthread -lfftw3 -lpython2.7)
```