INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

RAPPORT DE PROJET PLURIDISCIPLINAIRE

4EII 2018-2019

Antoine Riotte

Camille Yver

Haobo Wang

Fabio Eid Morook

RAPPORT DE PROJET PLURIDISCIPLINAIRE

Sommaire :

Introduction

Le projet pluridisciplinaire de cette année se base sur la réalisation d'une prothèse de main. Pour des raisons de faisabilité, il ne sera étudié que la réalisation d'un doigt actionné, grâce à la récupération des signaux myoélectriques du bras.

L'intérêt de ce projet était de parvenir à joindre différentes connaissances acquises au cours de notre formation en un seul projet, nous préparant ainsi davantage aux travaux que nous aurons à mener en entreprise.

Les compétences apportées au projet tenaient principalement de l'électronique et du microcontrôleur. Les objectifs annoncés en début de projet étaient ainsi la réalisation d'une carte électronique permettant de filtrer et amplifier un signal électrique reçu via des électrodes, ainsi que l'acquisition numérique des données, la transmission Bluetooth, et le pilotage d'un servo moteur.

Afin de parvenir à mener à bien ce projet, nous nous sommes basés sur la méthode scrum pour construire notre méthodologie. De plus, chaque bloc a été découpée en tâches précises, attribuées à une personne, et planifiées.

1 - Méthodologie

Pour un projet sur plusieurs mois et au sein d'une équipe de plusieurs membres, il est nécessaire d'avoir une méthodologie claire afin de parvenir à bien s'organiser au niveau du groupe, et d'avoir tous la même vision du projet

1.1 - Découpage du projet

Tout d'abord, la division du projet en bloc relativement indépendant était primordiale. Cela permet de répartir le travail équitablement entre les différents membres du groupe. Nous avons donc découpé le projet en 5 parties :

1) Système d'amplification et filtrage
2) Communication Bluetooth
3) Pilotage du servo moteur
4) Lecture des signaux sur la carte Nucléo
5) Alimentation

Ce découpage peut aussi mieux se comprendre sous forme de schéma :

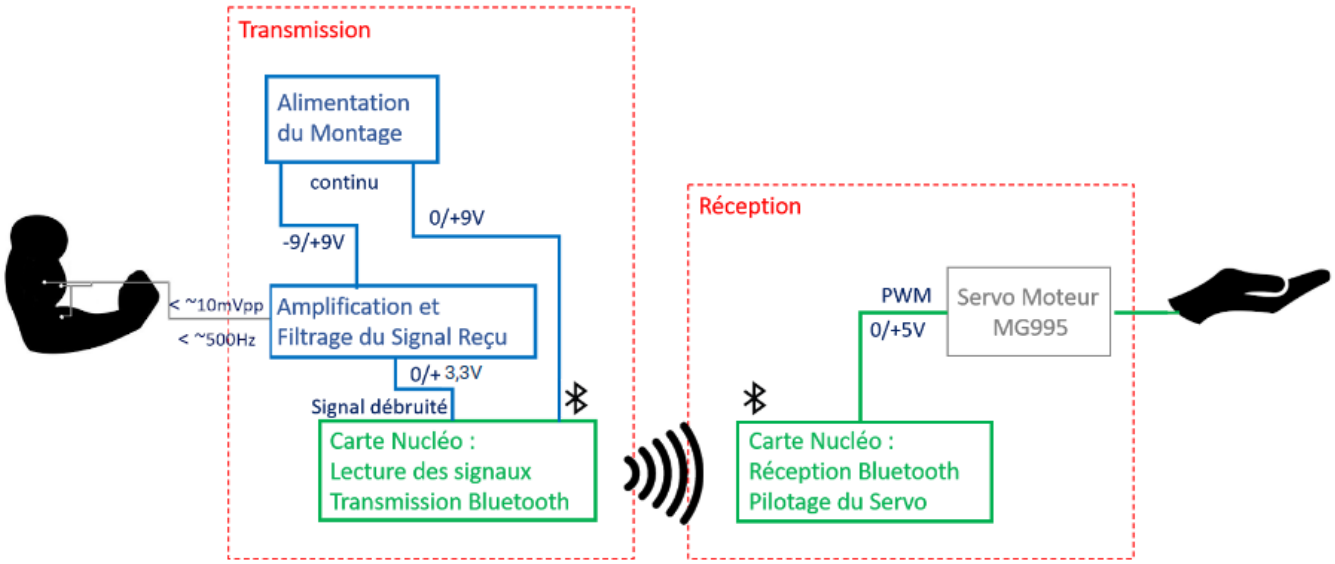


Figure 1- Schéma de découpage du projet

Sur ce schéma il est possible d'observer clairement les liens entre chaque tâche ainsi que leur rôle au sein du projet, il nous a permis de décrire pour chaque tâche sa fonction :

Système d'amplification et filtrage

Le système d'amplification et de filtrage s'occupera dans un premier temps d'amplifier le signal provenant des capteurs, puis de le filtrer pour en extraire l'information.

Ensuite, il sera nécessaire d'adapter le niveau du signal pour qu'il soit lisible sur une broche du microcontrôleur. Pour cela il faut garder un signal de fréquence relativement faible et de niveau inférieur à 0-3,3V.

En résumé :

Entrée : 10 mVpp - 500Hz bruité

Sortie : 0-3,3V de faible fréquence

Communication bluetooth

La communication entre les deux cartes Nucléo des blocs récepteur et transmetteur se fera par Bluetooth. Le programme des cartes Bluetooth sera fait à l'aide du logiciel STM32Cube avec le paquet d'expansion X-Cube-BLE1.

Pilotage du Servo moteur

Le bloc concernant le pilotage du servo-moteur sera piloté depuis la carte Nucléo, à l'aide du logiciel Keil.

Le principe sera d'activer le servo pour qu'il se rende à une certaine position quand le signal reçu est haut. Quand le signal est bas, le servo reprendra sa position initiale.

Lecture des signaux sur la carte Nucléo

Une entrée ADC du microcontrôleur sera configurée pour lire le signal entrant. En fonction de la valeur, par rapport à un seuil, une commande pour ouvrir ou fermer le doigt sera envoyé.

Alimentation

Le module d'alimentation permet d'adapter les niveaux des tensions aux besoins des différents composants utilisés. La partie électronique analogique nécessite une alimentation -/+9V par exemple.

1.2 - Outils

Pour mener à bien ce projet nous avons à notre disposition les ressources matériels et connaissances acquises. Nous utiliserons les logiciels STM32CubeMX ainsi que Keil uVision 5 pour la programmation des microcontrôleurs.

Pour l'élaboration de l'électronique analogique et la simulation du projet nous utiliserons le logiciel Orcad PSpice.

Les cours et TPs précédemment effectués lors de notre formation seront d'une grande utilité pour la réalisation du projet, notamment les cours et TP de microcontrôleur ainsi que les connaissances acquises en cours d'électroniques.

Pour notre gestion de projet nous avons réalisé un diagramme de Gantt détaillant les tâches à effectuer et leurs répartitions dans le groupe (voir Annexe 3).

Le Gantt réalisé initialement n'a pas été complètement respecté. Ceci est notamment dû au fait que les découpages étaient trop grossiers. Les tâches n'étaient pas assez détaillées, ce qui ne permettait pas vraiment de se rendre compte de la durée qu'elles pourraient prendre. Cela a été amélioré lors de la conception du Gantt réel.

D'autres outils sont à notre disposition, pour la communication et le partage de fichier : Messenger et Drive.

1.3 - Organisation

Pour ce projet il a été préconisé d'utiliser la méthode scrum, se basant sur des sprints d'une durée fixe. A la fin de chaque sprint, une étape du projet doit être finie. Nous avons tenté de tenir à ce format, cependant, l'espacement entre les séances et les quelques problèmes de découpage du projet au début ont rendu cette tâche compliquée.

Cependant nous avons gardé un rythme de réunion toutes les deux semaines afin de se tenir informé de l'état d'avancement de chaque partie du projet.

De plus une conversation de groupe sur Messenger nous permettait de rester en contact. Ainsi quand une question faisait surface, il était possible d'en discuter entre nous, sans pour autant se rencontrer physiquement. Cela nous a permis d'être flexible dans notre façon de gérer le projet. Cela a aussi permis d'éviter d'éventuels problèmes de communication.

2 - Développement

2.1 - Bloc Electronique

Système d'amplification et filtrage

Le signal perçu par l'électrode de surface peut facilement être perturbé par le bruit présent dans l'environnement.

Le signal SMEG en sortie des électrodes a une tension d'amplitude très faible de quelque microvolt a 4-5 mV. Le gain à prévoir est d'environ de 2000-20000, pour obtenir une tension de sortie entre 0V et 3.3V. Un potentiomètre ajouté au niveau du montage d'amplification permet de régler ce gain.

Pour enlever les signaux au-dessus de 1kHz et en dessous de 10 Hz, deux filtres sont faits dans le montage. Pour améliorer le filtrage, les deux filtres sont du second ordre.

Amplification :

Une pré amplification est placée en premier dans le montage avec un gain de 50. L'amplificateur d'instrumentation choisi est le AD621 qui a une haute impédance d'entrée et une large plage d'alimentation de 2.3V à 18V. Ce montage permet d'amplifier des signaux de très faibles amplitudes.

On utilise ensuite une amplification à l'aide d'un montage amplificateur non inverseur.

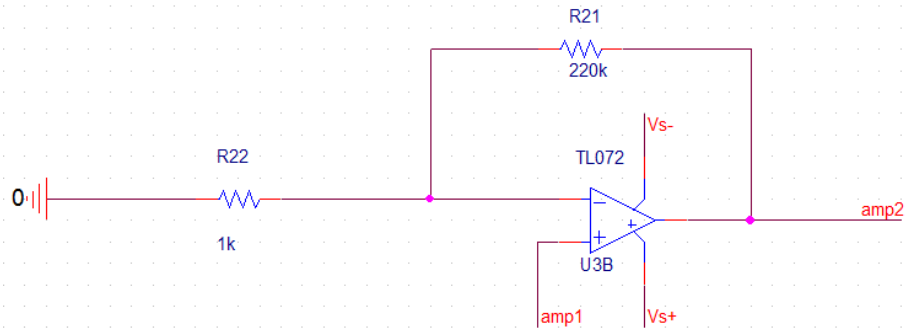


Figure 2- Amplificateur non-inverseur, 2e étape du système d'amplification

Afin de modifier le gain facilement, R21 est remplacé par un potentiomètre. Un potentiomètre linéaire est préférable dans notre cas pour régler une valeur précise mais ceux utilisés sont des potentiomètres logarithmiques, choisis car ils étaient disponibles et pour pouvoir régler le gain simplement et ainsi s'adapter aux différentes personnes et placement d'électrodes.

Filtrage :

Pour la partie de filtrage, le signal comportant les informations utiles se situe en fréquence de 10Hz à 1kHz. Un filtre passe-bas est alors utilisé avec fc=1kHz et un filtre passe-haut de fc=10Hz.

$f_1 = \frac{1}{2\pi RC} = 1kHz$ avec C=100nF => R1=1591.5Ω
$f_2 = \frac{1}{2\pi RC} = 10Hz$ avec C=100nF => R2=159 154Ω

Pour s'adapter aux valeurs qu'on peut trouver dans la série E12:

R1=2.2KΩ et R2=100k

Donc notre f1 et f2:

$$f_1 = \frac{1}{2\pi RC} = 723Hz$$
$$f_2 = \frac{1}{2\pi RC} = 15Hz$$

Après dans la bande de fréquence autour de 50Hz, on observe toujours un bruit fondamental pour les signaux électroniques. Donc un filtre coupe bande de 50Hz est mis en place pour enlever le bruit de 50 Hz.

Une amélioration qui pourrait être apportées est de remplacer le filtre coupe bande autours du 50Hz par un filtre passe haut de fc=60 Hz qui nous permet d'enlever le filtre coupe haut à 10Hz également.

Traitement du signal après amplification et filtrage :

Un circuit détecteur d'enveloppe est placé dans le montage pour extraire l'enveloppe et ainsi obtenir un signal simplement traitable par la carte Nucléo. Concernant l'interface avec la carte Nucléo, il faut limiter la tension de sortie. Pour cela on place une diode Zener avec une tension seuil 3.3V en sortie du montage. Donc la sortie est alors toujours inférieure à 3.3V.

Afin d'isoler le fonctionnement du circuit détecteur d'enveloppe de la sortie et de la diode Zener un AOP en montage non-inverseur fait la liaison ce qui permet à la fois de régler le niveau de tension de sortie.

Alimentation

La partie électronique analogique permettant l'amplification et le filtrage est alimenté et nécessite une alimentation -/+9V. L'alimentation du microcontrôleur nécessite une alimentation en 0/3.3V, la carte nucléo possède un régulateur intégré on peut donc utiliser directement le pin Vin pour une alimentation 9V.

2.2 - Bloc Microcontrôleur

Le second bloc concerne le traitement numérique de l'information, c'est à dire l'acquisition par des données sur une carte nucléo, l'envoie par Bluetooth, puis le pilotage du servo moteur. Les tâches seront présentées dans cet ordre afin de permettre une meilleure compréhension de cette partie, plutôt que chronologiquement.

Lecture des signaux sur la carte Nucléo

Il a été mis à disposition deux microcontrôleurs STM32 sur des cartes de développements nucléos. Ces cartes possédant plusieurs pin ADC, cette méthode a été retenue pour récupérer l'information. Cela nous a semblé la plus évidente et la plus simple.

L'écriture du code n'a pas été une étape compliquée. En effet, la configuration via CubeMX a permis une écriture automatique d'un certain nombre de paramètre. Il a été choisi pour cette acquisition une résolution de 12 bits, par exemple. Comme il n'est nécessaire de n'effectuer qu'une conversion, les autres paramètres sont laissés par défaut :

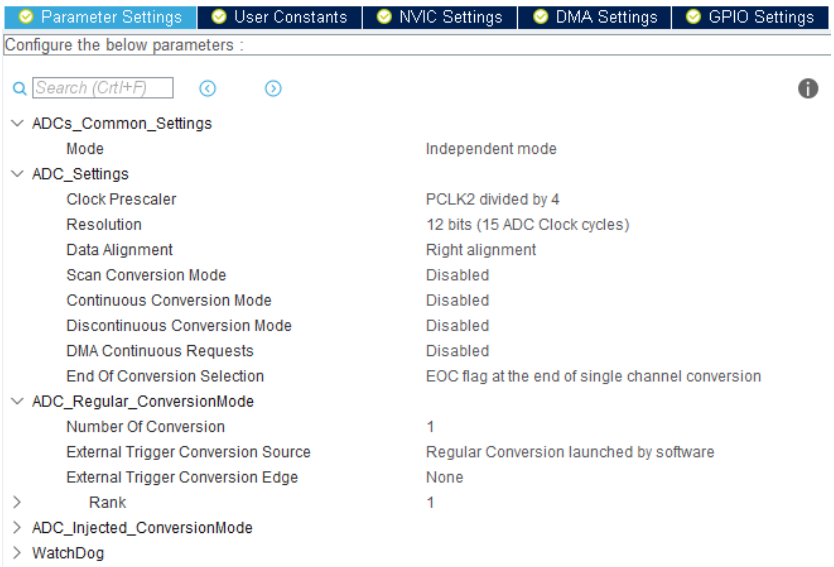


Figure 3 - Configuration de l'ADC sur CubeMX

De même nous avons décidé de n'activer la conversion que par appel d'une fonction *adc(),* créer par la suite. Une seule conversion sera donc réalisée par appel de cette fonction.

Après avoir généré le code, il va être possible d'implémenter notre fonction permettant l'acquisition de données. Le code est relativement court, puisqu'il est possible de faire appel aux fonctions fournies par la bibliothèque HAL_ADC.

Le but de cette fonction *adc()* est de prendre la valeur du signal arrivant sur la pin choisie (ici la A4), puis de vérifier si la valeur est supérieure ou inférieure à un certain seuil. Elle retourne ensuite la valeur '1' ou '0'.

Ainsi les fonctions HAL_ADC_Start et HAL_ADC_getValue sont utilisées pour récupérer une donnée :

Une fois la donnée récupérée, nous avons décidé de la normaliser. Cela permet de choisir le seuil plus facilement. La fonction de normalisation s'écrit :

$$RealV = 3{,}3 * \frac{ADCvalue}{4095}$$

Ici 3,3 correspond à la valeur maximale reçue et 4095 à la résolution de l'adc qui est de 12 bits. Dans le code la valeur a été multipliée par 10 pour ne pas avoir à gérer les nombres décimaux.

Le seuil a été décidé vers la fin du projet. En effet, il était nécessaire d'avoir des informations sur le signal envoyé depuis la carte du bloc électronique. Etant donné que ce signal passait de 0 à 3,3V systématiquement, le seuil a finalement été fixé à 1.

Ensuite il a fallu tester cette fonction. Tout d'abord avec une simple lecture d'un signal carré, de fréquence 500mHz, et d'une tension variant entre 0 et 3V. Cela permettait d'allumer une led en fonction de l'état du signal, haut ou bas. Le test a été concluant rapidement.

Communication Bluetooth

Pour la communication entre les deux cartes Nucléos, des modules "Bluetooth Low Energy (BLE)" compatibles avec les microcontrôleurs STM32 seront utilisés. La communication entre les deux cartes Nucléos a été faite avec l'aide de quelques fonctions qui étaient déjà implémentées dans l'API du BLE. Pour faire notre programme nous avons étudié un code fourni dans la documentation du Bluetooth Low Energy en ligne.

La première partie est l'initialisation du programme, cette partie est très importante car elle exécute trois configurations : La configuration du port COM, la configuration du bouton GPIO et la configuration de la LED comme GPIO.

Ensuite pour la communication entre les deux cartes, le programme lance une fonction dans la boucle while(1) du *main.c*. La connexion est faite grâce à la fonction *Make_Connection()*. Pour établir la communication entre les deux cartes, le programme crée la connexion avec la fonction de l'API *aci_gap_create_connection* dans la carte client. La carte serveur est quant à elle configurée pour recevoir des signaux avec la fonction *aci_gap_set_discoverable*.

Une fois la connexion établie la carte client attend des changements dans les valeurs du vecteur des informations, quand les changements ont lieus elle fait la lecture de ces valeurs.

Pour établir la communication entre les deux cartes il faut appuyer sur les boutons de redémarrage des cartes nucléos, la valeur du bouton change et le programme envoi de nouvelles valeurs. Il fait la communication des cartes, parce qu'on force la carte client à faire une lecture de donnée.

Pour vérifier la communication il faut que la LED2 des deux cartes s'éteignent après une courte période d'allumage. Après l'établissement de la communication, la carte client attend le changement de valeur pour lire des données, cette partie du changement des données est codée dans la fonction *Attribute_Modified_CB*.

Enfin le test pour assurer le bon fonctionnement du code a été fait avec l'aide des LEDs des cartes nucléos. Nous avons fait le test d'allumer/éteindre la LED d'une carte en appuyant sur le bouton de l'autre carte. (Dans le code il vérifie le changement d'état du bouton et envoie un signal pour allumer/éteindre la LED).

Pilotage du Servo moteur

La dernière tâche de la partie microcontrôleur concerne le pilotage du servo moteur. Il a été mis à disposition un doigt imprimé en 3D, qui s'actionne à l'aide d'une ficelle attachée à un servo moteur. Le servo moteur (MG995) est un moteur asservi en position. La plage d'alimentation s'étend de 4,8 V à 7,2 V. Il se pilote en position à l'aide d'une consigne (un signal PWM, de fréquence 50Hz). Il permet une rotation de 120°.

Comme pour l'ADC, l'utilisation de CubeMX permet une configuration plus rapide du code. Une PWM est générée à l'aide d'un timer (ici timer1). Une option spéciale sur CubeMX permet de fournir une PWM en sortie. La configuration de la clock est ici aussi très importante, il faut une fréquence de PWM de 50Hz, pour répondre aux besoins du servo moteur. Il est aussi nécessaire de définir le prescaler et la période du compteur. La période est fixée à 200 pas, cela rend les calculs plus faciles puisque la période et de 20ms, la fréquence interne du timer de 84MHz, il possible ensuite de retrouver le prescaler tel que :

$$prescaler = \frac{fréquence\ timer}{fréquence\ PWM * fréquence\ compteur\ PWM} - 1 = \frac{84.10^6}{50 * 200} - 1 = 8399$$

La configuration donne donc :

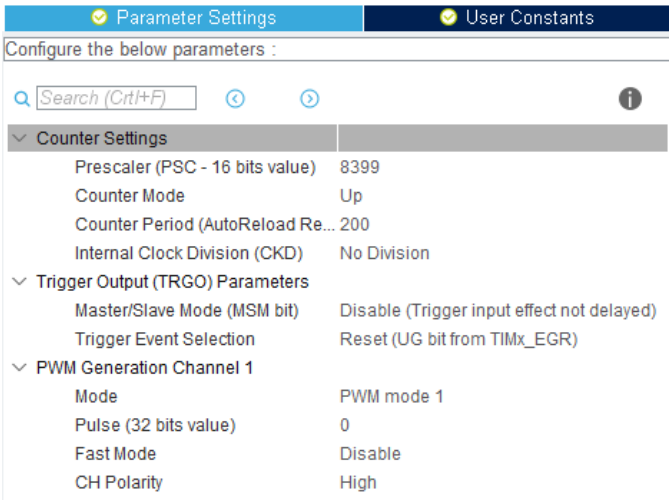


Figure 4- Configuration du timer2 pour la PWM sur CubeMX

Une fois le code généré, Il faut écrire les fonctions permettant la rotation du servo moteur. Celui-ci aura à prendre deux positions : une pour le doigt ouvert et une pour le doigt fermé. Le moteur tournant de 120° maximum, il nous faudra donc une consigne allant de 0° à 120°. Voici le schéma de fonctionnement de la consigne d'un servo moteur :

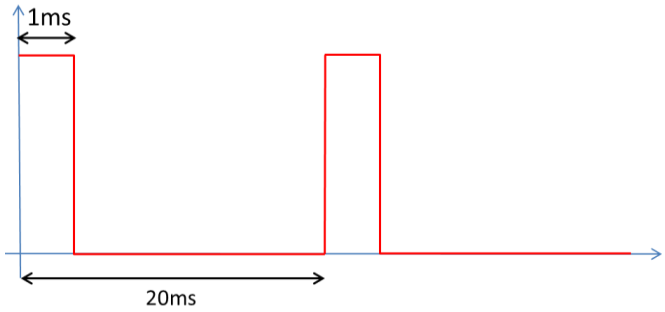


Figure 5- Consigne d'une PWM pour servo moteur

Ici pour une consigne de 1ms, la position est de 0. Elle est de 120° pour 2ms. Notre servomoteur aura donc une consigne variant de 1ms à 2ms en fonction de la position d'ouverture du doigt.

La résolution étant de 200, 1ms de 20ms correspond à 10 pas de 200 pas. La consigne a envoyé au servo-moteur sera de 10 ou 20.

Cependant, nous nous sommes ici heurtés à un problème. La carte STM32 envoie une tension ne variant qu'entre 0 et 3,3V, or le moteur a besoin d'une tension de fonctionnement supérieur à 4,8V. Pour pallier ce problème, il a fallu ajouter une amplification (voir Annexe 4.8). Il a été choisi une amplification de 5V, grâce à un transistor NMOS BS170 et une résistance de $10k\Omega$. Ce montage est inverseur, ce qui va imposer une petite modification à notre consigne :

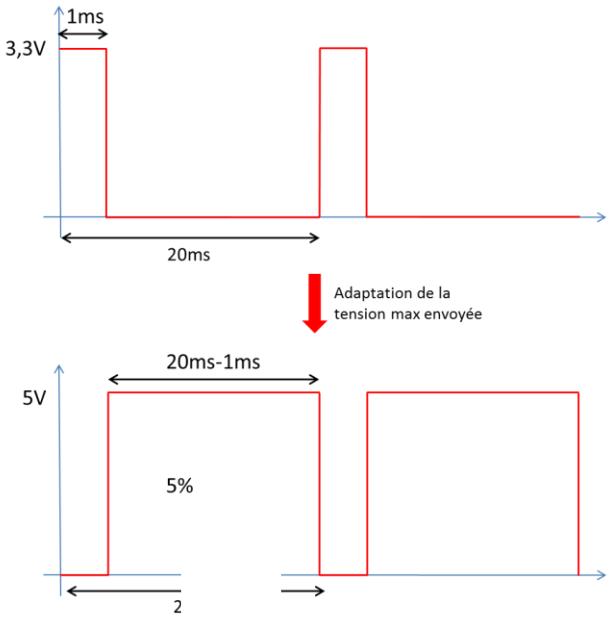


Figure 6 - Consigne pour PWM avant et après amplification

Donc dans le code, plutôt que d'avoir une consigne de 10, il y aura ici une consigne de 200-10.

Une fois ces modifications effectuées, les consignes sont implantées dans deux fonctions, une *ouvrir()* et une *fermer()*, selon que le doigt doit être ouvert (position 0, de consigne de 200-10) ou fermé (position 120 donc 200-20). (Voir Annexe 1)

Pour tester, il a suffi de coder dans le *while(1)* du *main.c*, les fonctions *ouvrir()* et *fermer()* séparées par un délai. Ainsi nous avons pu vérifier la rotation du servo moteur.

Ces fonctions ne prennent aucun paramètre afin qu'il soit plus facile de les utiliser par la suite, avec l'intégration du Bluetooth.

Intégration du code

L'intégration du code a été une période relativement longue, par rapport à ce qui avait été prévue. En effet le Bluetooth a un code très complexe, et il a été difficile d'y incorporer d'autres éléments.

L'un des tests effectués sans problème était celui de l'intégration de l'ADC et de la PWM. Mis sur une seule carte les deux codes fonctionnent sans modifications nécessaires : en fonction de l'entrée (toujours un signal carré de fréquence 500mHZ provenant d'un GBF), le doigt se plie et se déplie.

Le code est assez simple, il suffit de récupérer la valeur de *adc()* et, en fonction de la valeur, de lever ou baisser le doigt.

Cependant l'intégration entre La PWM et le Bluetooth ou l'ADC et le Bluetooth a été plus long. Cela est dû en partie au fait que les pins utilisés initialement par le Bluetooth et les deux autres parties, étaient les mêmes. Or n'ayant pas configuré nous-même le Bluetooth, cela n'est pas apparu évident à nos yeux au premier abord.

Nous avons pu sans trop de difficultés changer les pins de l'ADC et de la PWM. Du fait de dysfonctionnements de notre version de CubeMX, il n'a pas été possible de modifier directement le projet Bluetooth. Il a donc fallu modifier nos projets ADC et PWM à part et incorporer à la main tout le code et les configurations dans le projet Bluetooth, sur Keil.

Ensuite, pour la "PWM + Bluetooth", il a d'abord été fait un test avec un bouton : En appuyant sur un bouton d'une carte, le signal est envoyé par Bluetooth sur l'autre carte. Une fois le signal envoyé, dans la fonction *receive_data*, les fonctions *ouvrir()* et *fermer()* sont réalisées. Ce test s'est relativement bien passé et n'a pas pris beaucoup de temps.

Pour la partie "ADC + Bluetooth", le but était d'allumer une led sur une carte, à partir des données reçues sur l'autre carte. Celles-ci sont obtenues une fois encore par un signal carré venu du GBF. La fonction *adc()* est appelée avant les fonctions d'envois du Bluetooth et sa valeur est insérée

dans la première case du tableau envoyé en paramètre. Ensuite, selon la valeur, la led 2 est allumée ou éteinte. Ce test s'est aussi bien déroulé.

Enfin pour l'intégration du code complet ("Bluetooth + ADC + PWM"), le but était de faire ouvrir ou fermer le doigt à partir de signal carré créé par le GBF.

Une fois la communication établie entre les deux cartes, la carte client attend un changement d'une valeur dans le vecteur transmis entre les deux cartes. Après le test de connexion, la fonction *adc()* fait la lecture de la valeur reçue du GBF, puis la fonction *user_process* l'envoie vers la carte client (dans la première case du tableau envoyé) via la fonction *send_data*. La carte client reçoit ce tableau via la fonction *receive_data* et ne fait la lecture que de la première case, il la teste ensuite. Si cette valeur est "1" le doigt se ferme, et si cette valeur est "0" le doigt s'ouvre.

Après quelques problèmes de perte de connexion, dus entre autres aux conditions d'envoi du *user_process* le test a fini par fonctionner. Une légère modification des conditions, l'ajout d'un *if* testant la valeur du tableau, avec le test de la connexion, a permis de régler ce problème.

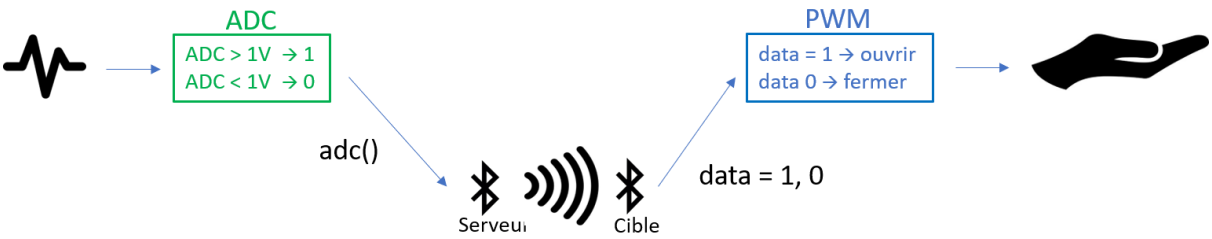


Figure 7- Schéma de fonctionnement du bloc microcontrôleur

3 – Résultat

3.1 - Problèmes rencontrés

Ce projet aura posé quelques problèmes légers, cependant tous ont pu être réglés avant la fin des séances.

Dans le bloc électronique, les problèmes sont principalement arrivés au début du projet, un premier système a été conçu d'un seul bloc ce qui rendaient les modifications complexes. Nous sommes donc repris les choses à la base, montage par montage, afin de pouvoir valider certaines parties (en pratique ainsi qu'en simulation). Puis passer au circuit d'après.

Dans le bloc microcontrôleur au contraire, les problèmes sont survenus vers la fin du projet, au moment de l'intégration. Le choix des pins à utiliser et la continuité de la connexion Bluetooth lors de l'intégration des trois parties ont posés quelques difficultés.

3.2 - Résultat du système d'amplification et filtrage

Le montage est d'abord validé en simulation et nous avons bien une sortie de tension qui sature à 3.3V. (la courbe violette dans la figure suivante)

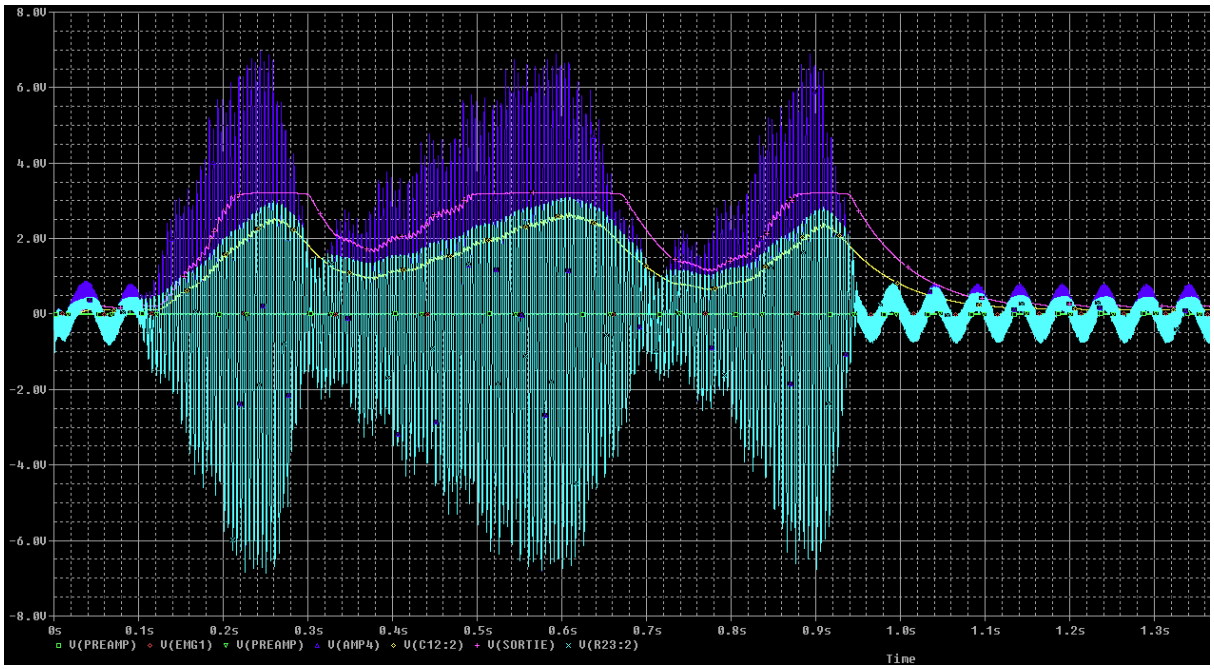


Figure 8 - Résultat de simulation du montage complet (en Rouge signal en entrée du microcontrôleur, en jaune le signal avant l'adaptation de niveau, en bleu et cyan différentes étapes d'amplification)

Ensuite nous faisons le câblage pour la partie pratique et le teste avec un générateur qui va nous aider de générer le signal d'entré. Puis nous testons avec les électrodes sur notre camarade :

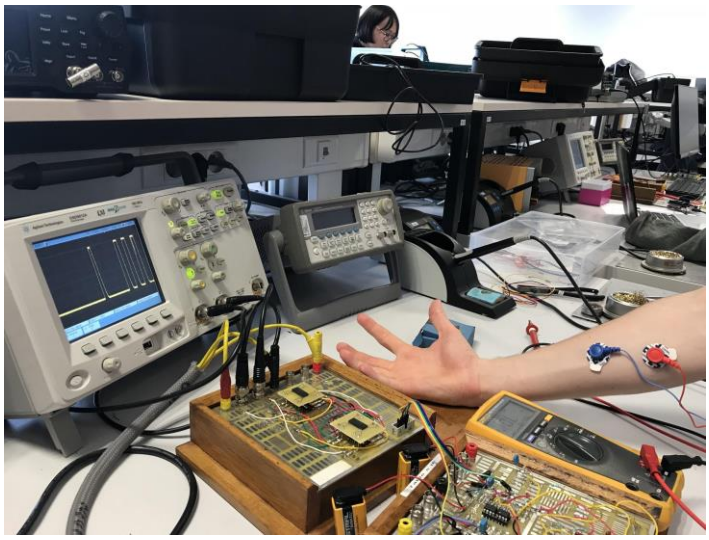


Figure 9 - Photographie montrant le système lors de l'expérimentation

L'amplitude de sortie peut être fixé par le potentiomètre. Le gain réglé, nous avons bien un signal qui sature à 3.3V en serrant et 0V en relâchant la main.

3.3 - Résultat de la communication entre les cartes et le servo moteur

Au final, la partie microcontrôleur a atteint son objectif : acquérir, transmettre et traiter une donnée, afin d'actionner une prothèse de doigt. Assemblé cela nous donne ce montage :

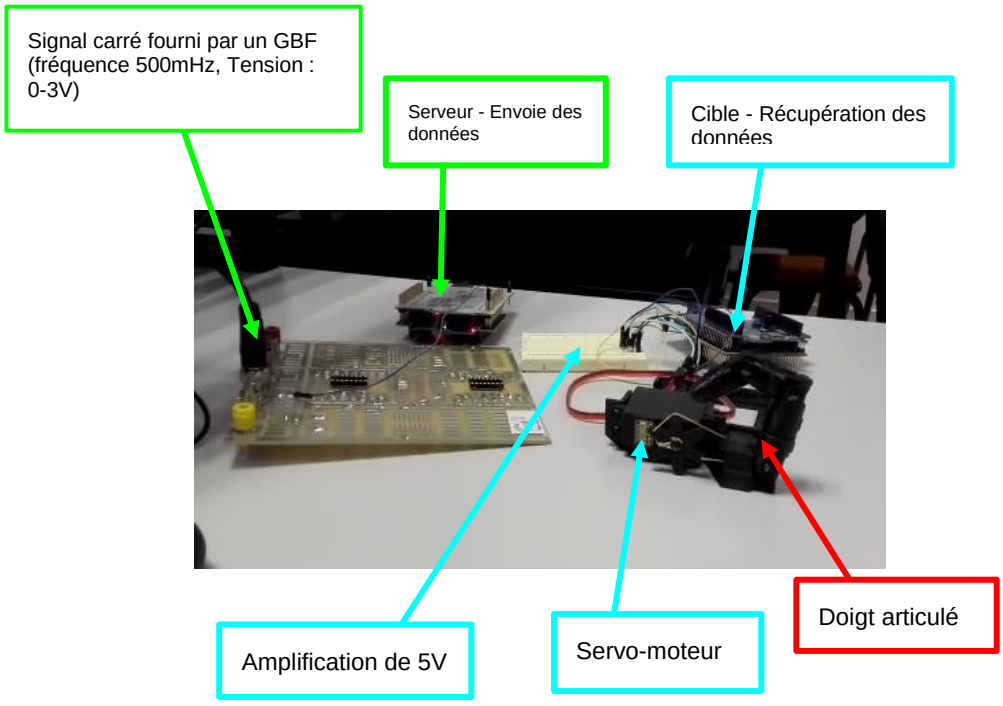


Figure 10 - Montage de la partie microcontrôleur

Il n'y a pas de problème à déclarer, le doigt se place en position fermée quand le signal est à l'état haut et en position ouverte quand le signal est à l'état bas. Cela correspond bien aux états donnés par la carte électronique en fonction de la fermeture ou de l'ouverture du poing.

3.4 - Résultat global

L'intégration finale n'a pas été un problème, elle s'est faite assez rapidement. Quelques modifications ont tout de même dû être ajoutées, comme par exemple le délai entre la réception et l'envoie des données sur la carte serveur. De même, l'alimentation du serveur est passé de l'ordinateur, avec une alimentation USB, à l'alimentation sur la carte électronique. L'acquisition des données n'était pas possible sinon. Nous avons également rajouté une led sur la carte électronique afin d'observer sans passer par l'oscilloscope, si le filtrage et l'amplification se faisaient correctement.

Nous avons cependant un léger problème d'acquisition du signal depuis les électrodes. En effet le fonctionnement de celles-ci semble aléatoire, et plusieurs essais semblent nécessaire avant de pouvoir constater le fonctionnement du projet dans son ensemble.

Conclusion

Ce projet a mis en œuvre les connaissances acquises lors de notre formation. Il aura aussi permis d'apprendre à gérer un travail de groupe sur plusieurs mois. Nous aurons eu à faire face à plusieurs difficultés, par exemple le fait de savoir reprendre une solution du début quand il est visible que les résultats ne sont pas satisfaisants, ou bien aborder par nous-même des solutions techniques telles que le Bluetooth. Enfin nous aurons su appliquer et comprendre les notions vues au cours de notre formation, avec le filtrage d'un signal en la mise en œuvre d'une PWM.

Table des Figures :

Tables des annexes :

ANNEXES

Annexe 1 - Code microcontrôleur commenté

```
102 void ouvrir(void){
103   htim2.Instance->CCR1=210-10;            //position 120 du servo
104 }
105
106
107 void fermer(void){
108     htim2.Instance->CCR1=210-20;          //position 0 du servo
109 }
110
111
112
113 uint8_t adc(void){
114     HAL_ADC_Start(&hadc1);                //on commence l'acquisition
115     adcValue = HAL_ADC_GetValue(&hadc1);  //on récupère la valeur
116
117     realV = 33 * adcValue / 4095;         //on normalise la valeur
118
119     if(realV > 10){                       //seuil à 10 --> 1V
120       BSP_LED_On(LED2);
121       return 1;                           //fermeture du doigt
122     }
123     else{                                 //ouverture du doigt
124       BSP_LED_Off(LED2);
125       return 0;
126     }
127   }
```

Fichier main.c, fonction adc, fermer et ouvrir

```
225 /**
226   * @brief  This function is used to receive data related to the sample service
227   *         (received over the air from the remote board).
228   * @param  data_buffer : pointer to store in received data
229   * @param  Nb_bytes : number of bytes to be received
230   * @retval None
231   */
232
233 void receiveData(uint8_t* data_buffer, uint8_t Nb_bytes)
234 {
235   if (data_buffer[0] == 1){
236     fermer();
237     BSP_LED_Toggle(LED2);
238   }else if(data_buffer[0] == 0){
239     ouvrir();
240     BSP_LED_Toggle(LED2);
241   }
242
243   for(int i = 0; i < Nb_bytes; i++) {
244     printf("%c", data_buffer[i]);
245   }
246 }
247
```

Fichier sample_app.c, fonction receiveData

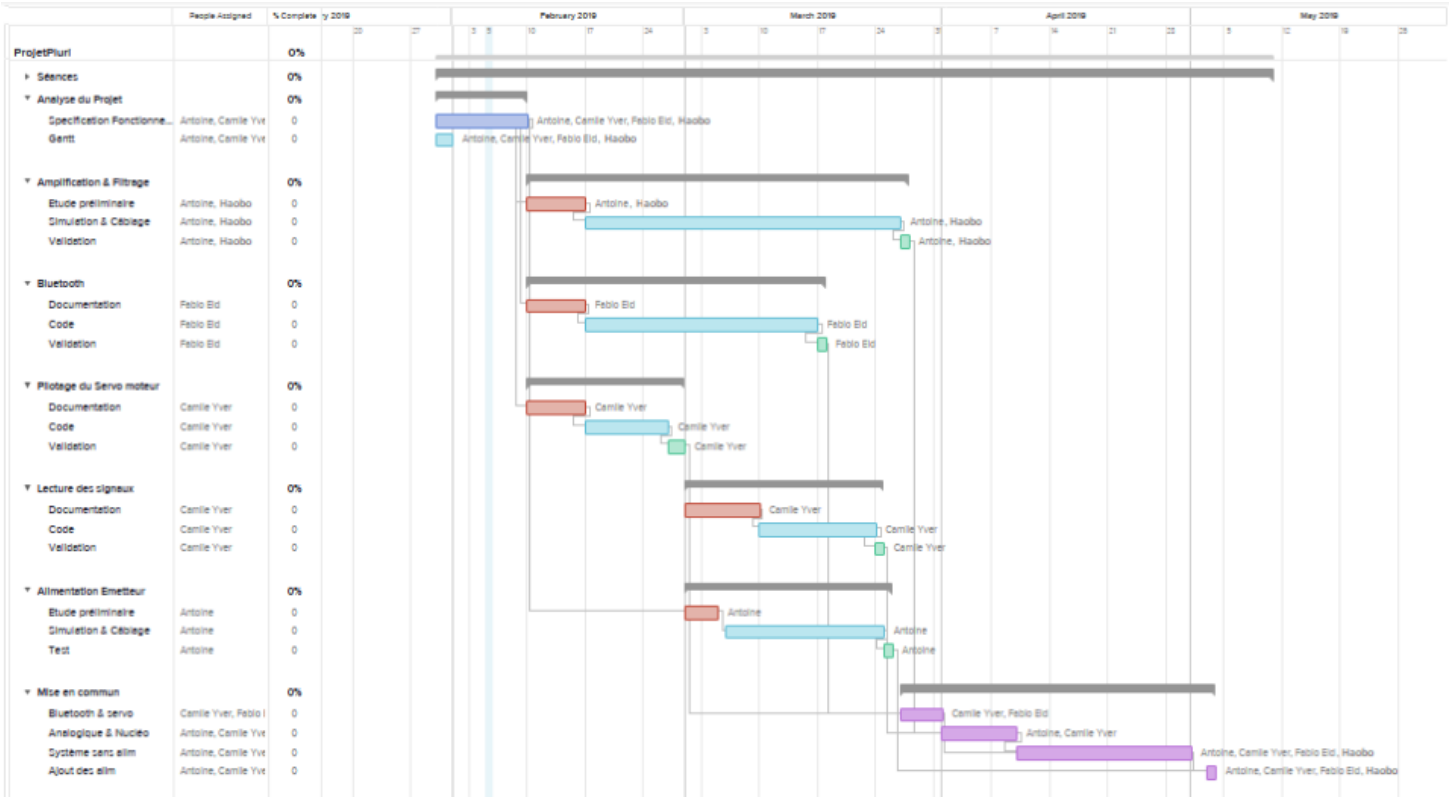
```
/**
  * @brief  Configure the device as Client or Server and manage the communication
  *         between a client and a server.
  * @param  None
  * @retval None
  */
static void User_Process(void)
{
  if (set_connectable)   {
    /* Establish connection with remote device */
    Make_Connection();
    set_connectable = FALSE;
    user_button_init_state = BSP_PB_GetState(BUTTON_KEY);  }
  if (BLE_Role == CLIENT) {
    /* Start TX handle Characteristic dynamic discovery if not yet done */
    if (connected && !end_read_tx_char_handle){
      startReadTXCharHandle();    }
    /* Start RX handle Characteristic dynamic discovery if not yet done */
    else if (connected && !end_read_rx_char_handle){
      startReadRXCharHandle();    }
    if (connected && end_read_tx_char_handle && end_read_rx_char_handle && !notification_enabled) {
      BSP_LED_Off(LED2);            //end of the connection and chars discovery phase
      enableNotification();
    }  }
  uint8_t data[20] = {0,'1','2','3','4','5','6','7','8','9','A','B','C','D','E','F','G','H','I','J'};
  if (data[0]==1 || data[0]==0)  {
    if(connected && notification_enabled){
      //Send a toggle command to the remote device
      data[0] = adc();              //Récupère la valeur de l'adc
      HAL_Delay(100);               //Attente 100ms pour temporiser l'envoie de donné
      sendData(data, sizeof(data)); //envoie de sonnée via bluetooth
    }
  }
```

Fichier app_x-cube-ble1.c, fonction User_Process

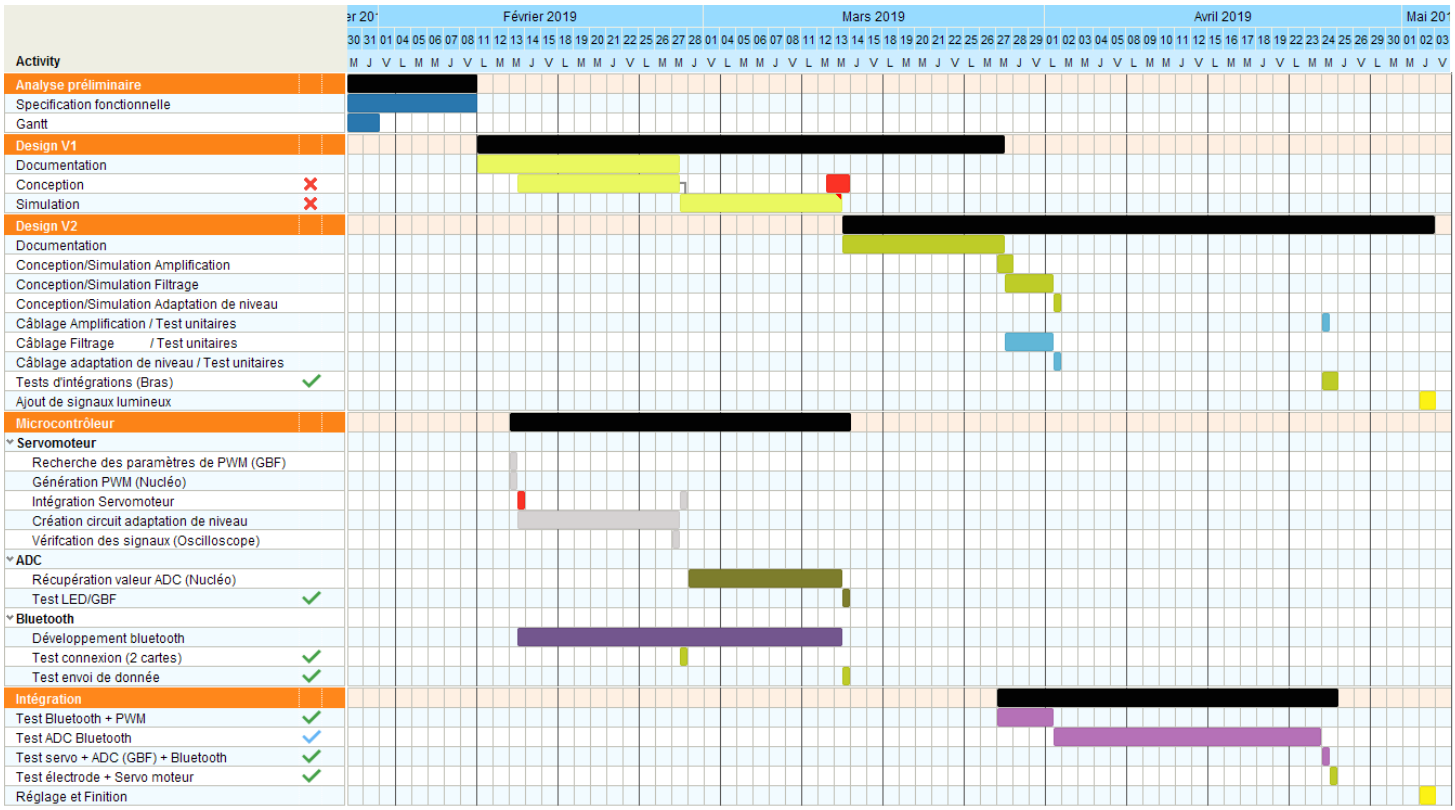
Annexe 2 - BOM

1. TL082 x5 (3x2)
2. AD621 x1
3. 1N5226BTR x1
4. D1N4148 x1
5. BS170 x1
6. Résistance :
    a. 1 kΩ x3
    b. 2.2 kΩ x2
    c. 22 kΩ x4
    d. 39 kΩ x1
    e. 68 kΩ x2
    f. 100 kΩ x3
    g. 220 kΩ x2
    h. 280 kΩ x1
    i. 330 kΩ x1
    j. 10 kΩ x1
7. Condensateur :
    a. 47 nF x2
    b. 100 nF x7
    c. 220 nF x1

Annexe 3 - Diagramme de Gantt



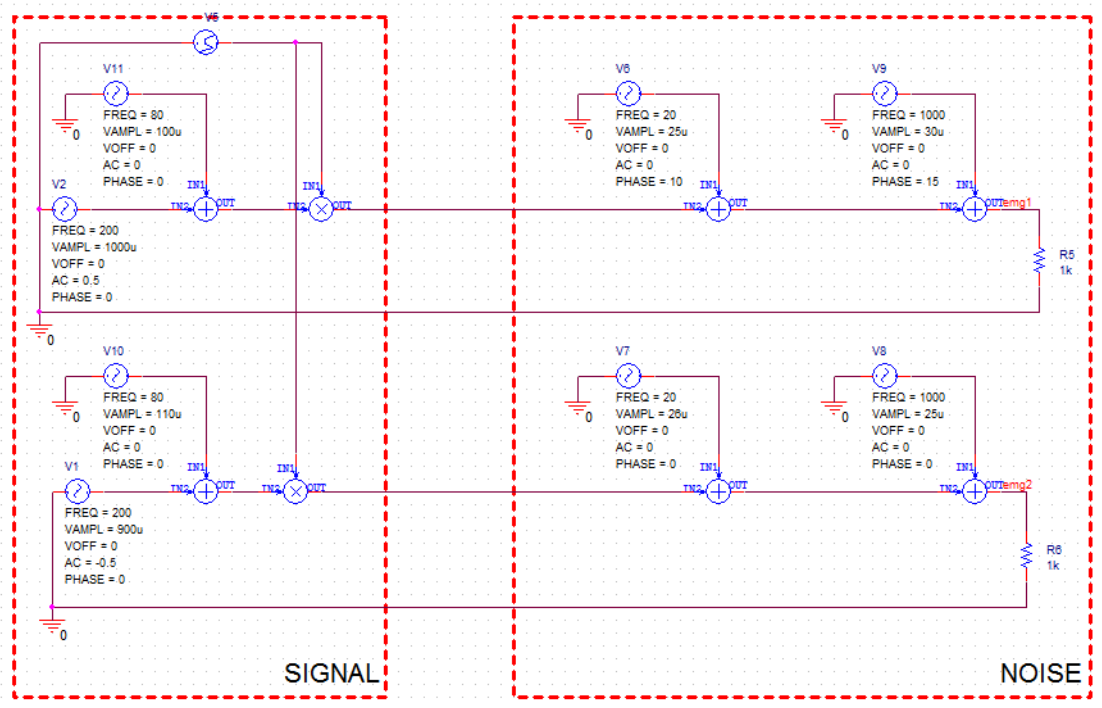
Gantt prévisionnel



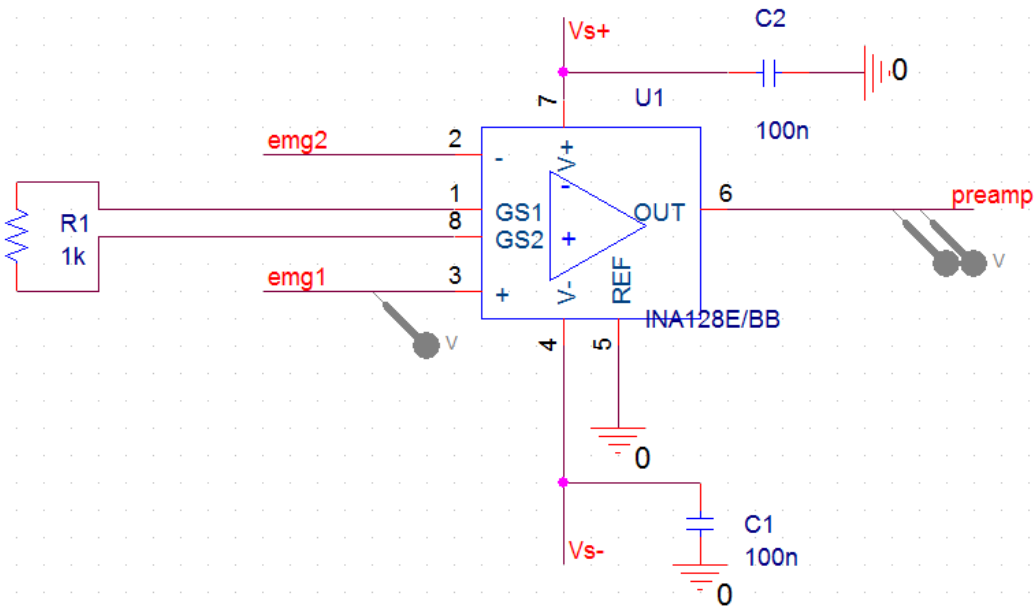
Gantt Réel

Annexe 4 - Schémas électroniques

0 - Source

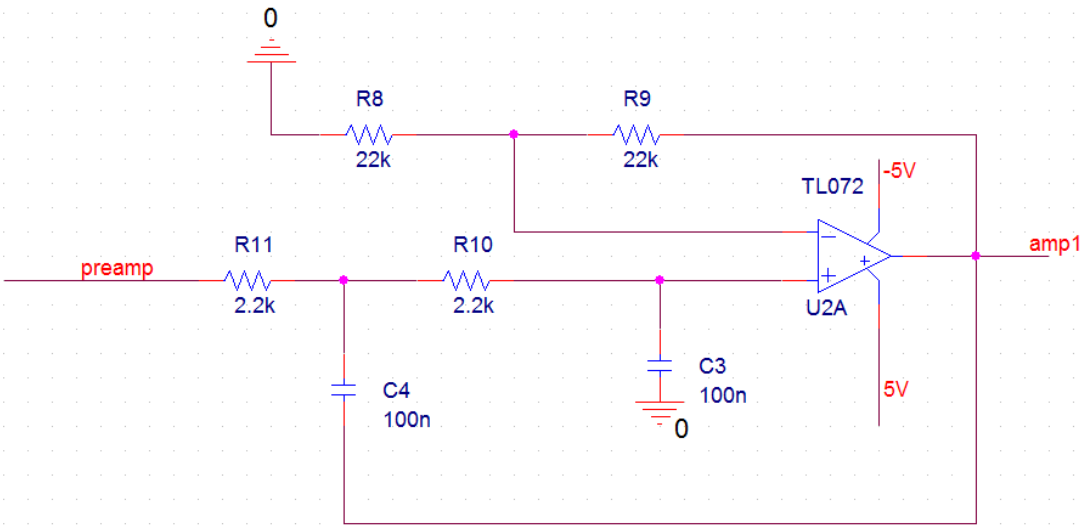


1 - Preamplication de gain de 50



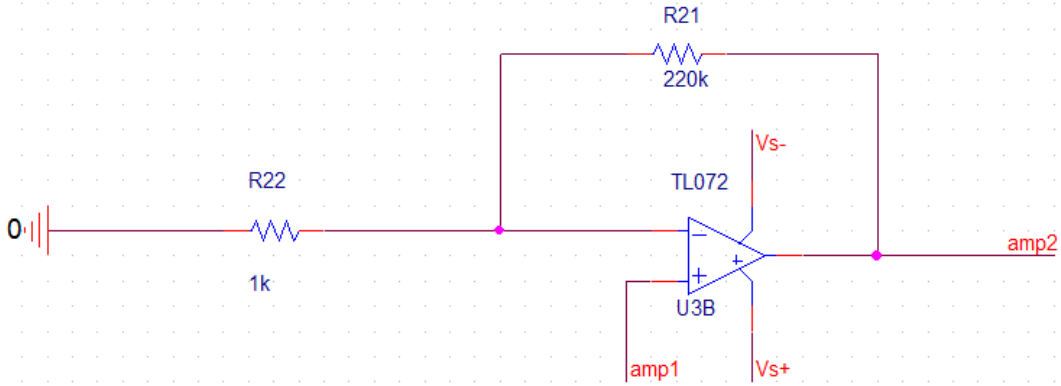
(Nous avons pris le AD621)

2 - Filtre passe bas fc=723 Hz avec un montage non inverseur

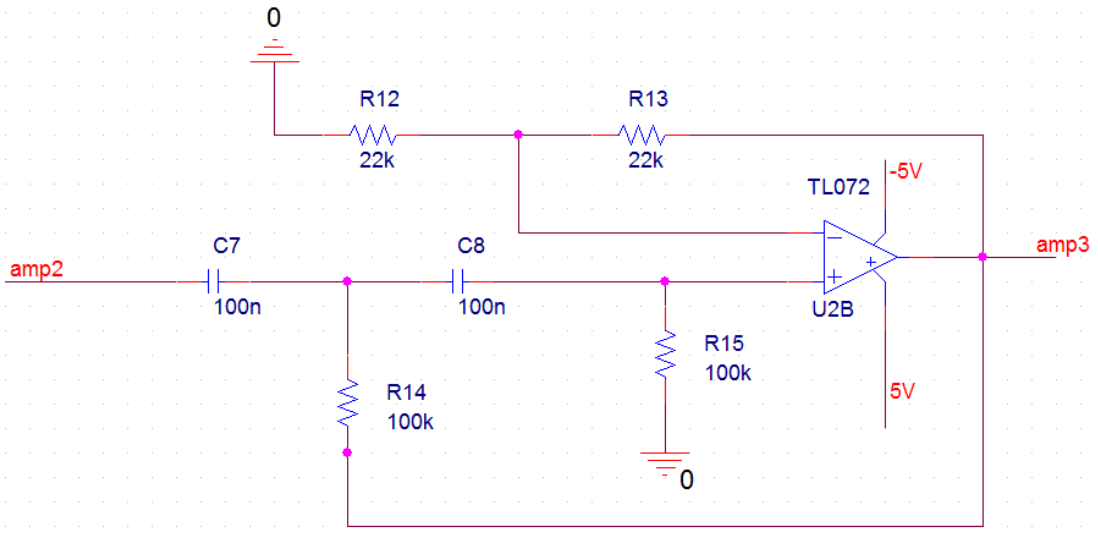


TL082 à la place de TL072

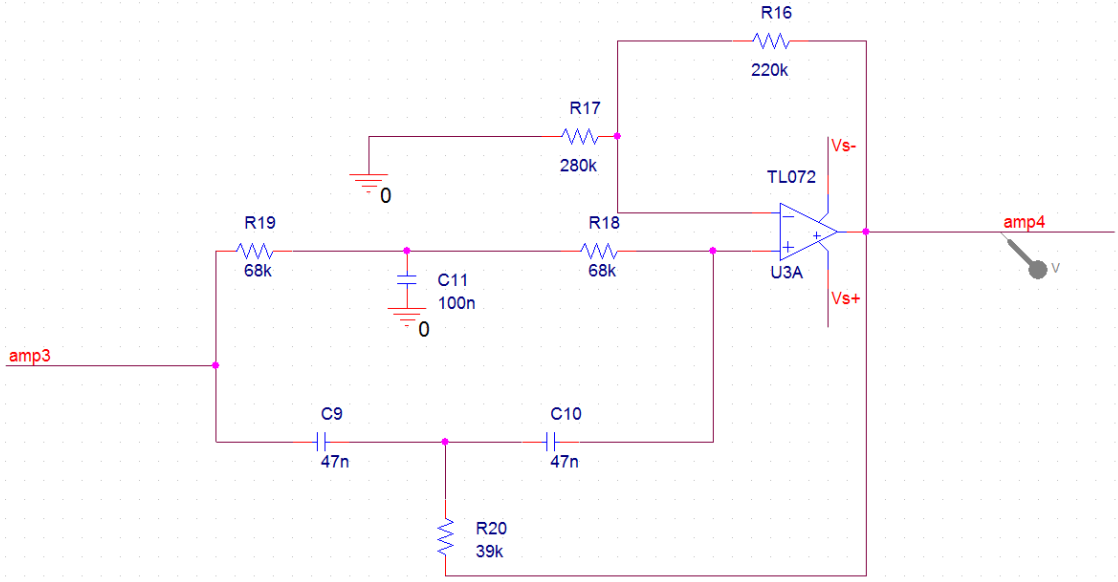
3 - Amplification de gain de 221



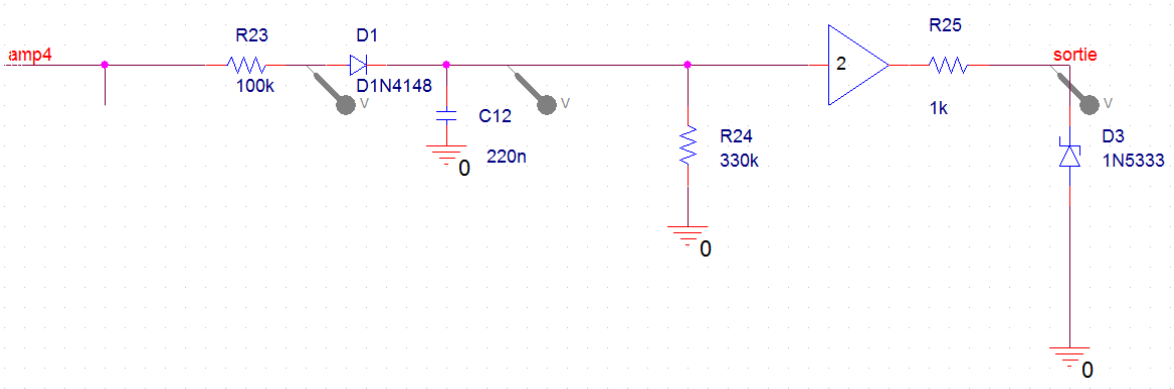
4 - Filtre passe haut avec fc=15Hz avec un montage non inverseur



5 - Filtre coupe bande avec fc=50Hz

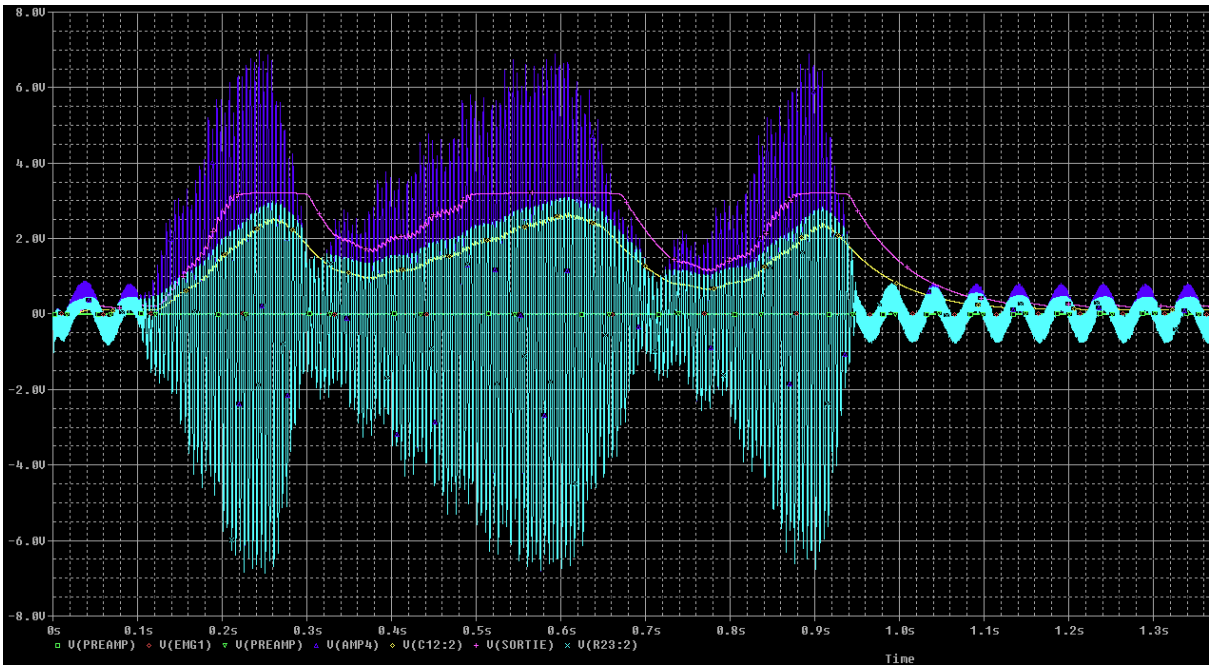


6 - Détection d'enveloppe pour le signal à la sortie, isolation avec un AOP et une restriction avec diode Zener

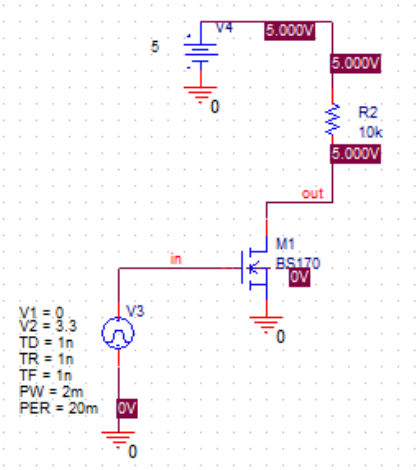


Diode Zener pour régler la tension :1N5226BTR

7 - Résultat de la simulation



8 – Amplification de 5V en entrée du servo-moteur :



INSA Rennes
20 Avenue des Buttes de Coësmes
CS 70839
35708 Rennes Cedex 7

Tél. +33 (0) 2 23 23 82 00
Fax +33 (0) 2 23 23 83 96

www.insa-rennes.fr