ROBOTIQUE MOBILE S8 – MEA4

Année scolaire 2020-2021

"Projet : Modélisation, conception et réalisation d'un robot nageur"

Etudiants : Anna AGOBIAN, Tatiana DEWILDEMAN, Théo LEMAIRE, Paul MASSON

Encadrants : René ZAPATA, Éric DUBREUIL, Lionel LAPIERRE

SOMMAIRE

1 INTRODUCTION

Ce document est un rapport du travail effectué dans le cadre des projets de robotiques mobiles au S8 en MEA4.

1.1 Contexte

Le but de ce projet est de développer un robot humanoïde nageur (crawl et/ou papillon). Celui-ci doit être capable de se déplacer et de s'orienter dans un environnement ouvert et agité comme la mer. Afin de s'orienter, celui-ci comportera une tête, déjà développée, utilisant un algorithme de perception des bouées jaunes.

1.2 Attentes

Modélisation, conception et réalisation d'un robot nageur. Le travail consiste à étudier les modèles de nage d'un robot nageur (crawl et/ou papillon), de réaliser des simulations (à partir d'outils existants), de réaliser une maquette de bras, d'effectuer une recherche internet sur des moteurs qui pourraient être adéquats et enfin, de réaliser un robot réel (échelle de 1/2 ou 1 à préciser).

2 MODELISATION

2.1 Adaptation du bras humain

Le bras humain est constitué de 5 degrés de liberté :

- Rotation principale de l'épaule a_1
- Écartement de l'épaule du corps a_2
- Rotation axiale de l'épaule a_3
- Pliement du coude a_4
- Rotation axiale de l'avant-bras a_5

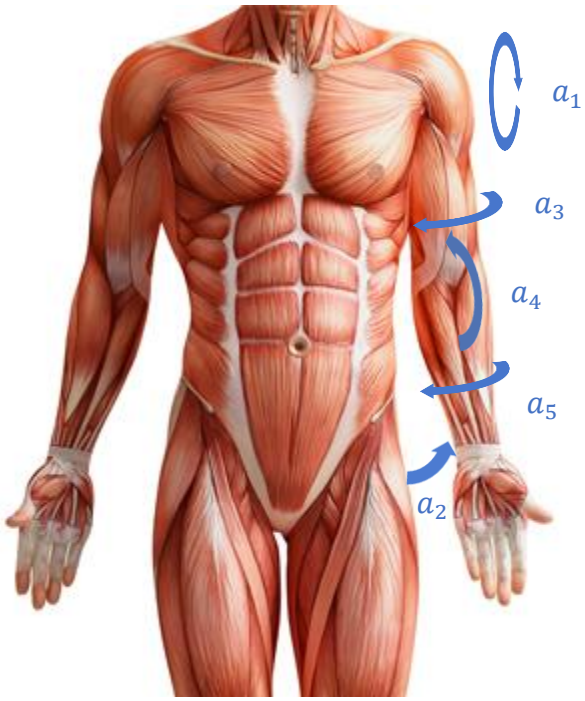


Figure 1: *Illustration des degrés de liberté du bras humain*

A chacun de ces degrés de liberté est associé une articulation, générant une complexité de modélisation non négligeable. Cependant, lors d'un mouvement de nage tel que le crawl, seuls les articulations a_1, a_3 et a_4 sont indispensables. Par soucis de simplicité, nous nous limiterons donc à ces 3 degrés de liberté.

2.2 Détermination du mouvement

Afin de déterminer le mouvement du bras lors de la nage du crawl, nous utilisons le simulateur *MATLAB*, nommé *KynProfile*, permettant de contrôler un bras simplifié aux 3 degrés de libertés précédemment évoqués.

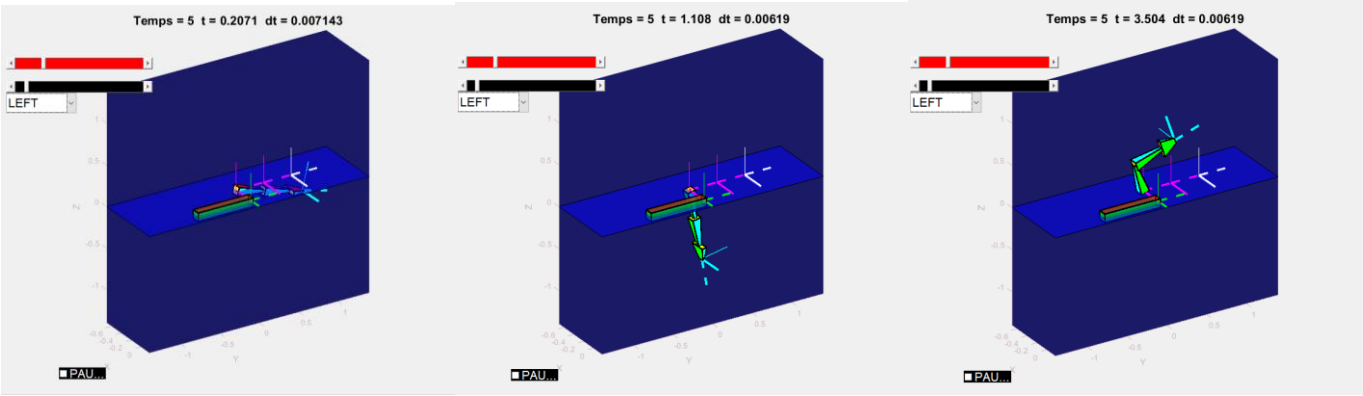


Figure 2: *Illustrations du simulateur KynProfile*

Ce simulateur prend en entrée les commandes angulaires de chaque articulation. Afin de former ces commandes, nous utilisons un 2$^{\text{ième}}$ simulateur, nommé *Deform_Splines*.

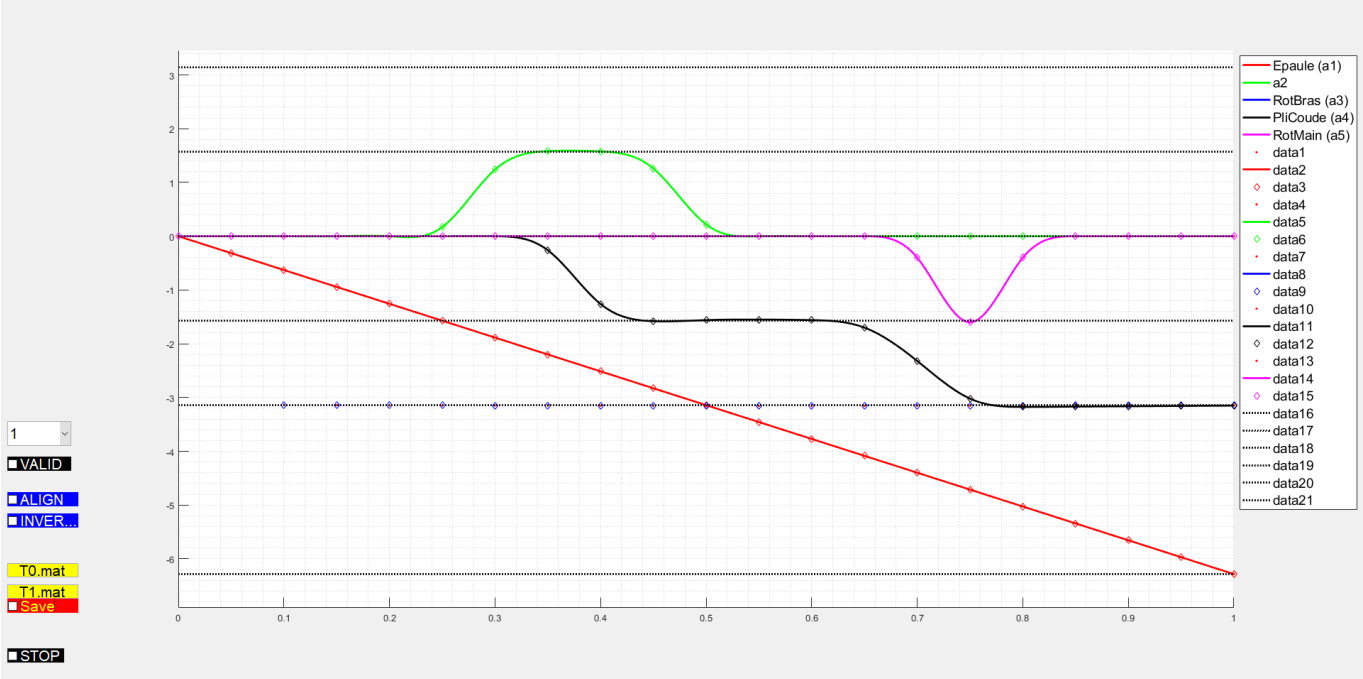


Figure 3: *Illustration du simulateur Deform_Splines*

Celui-ci nous permet de manier les trajectoires angulaires de chaque articulation. L'axe des abscisses représente une période, ici un mouvement de crawl, tandis que l'axe des ordonnées représente la position angulaire de l'articulation. Il est donc important que chacune des articulations en $x = 1$ se retrouve à sa position initiale $\pm 2\pi$ pour garantir la continuité du mouvement entre chaque période.

Grâce à ces 2 simulateurs, nous avons décomposé un mouvement de crawl en un enchaînement de mouvements élémentaires et faisables avec les 3 articulations :

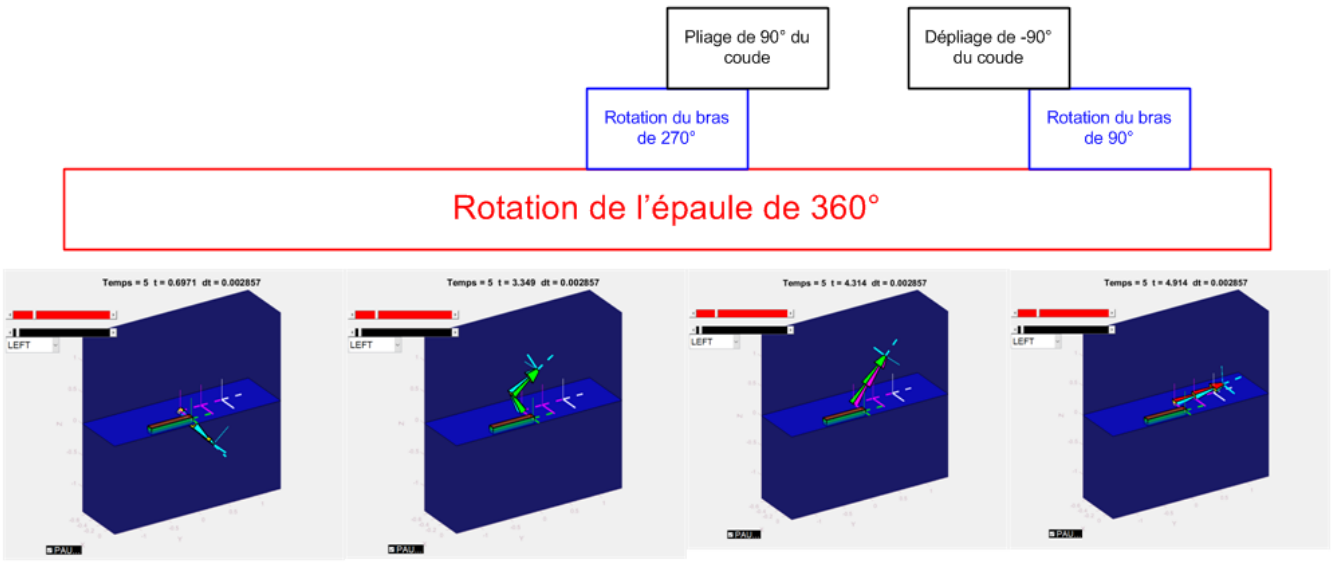


Figure 4: *Enchainement des mouvements lors du crawl*

Cet enchaînement se traduit par les trajectoires angulaires suivantes :

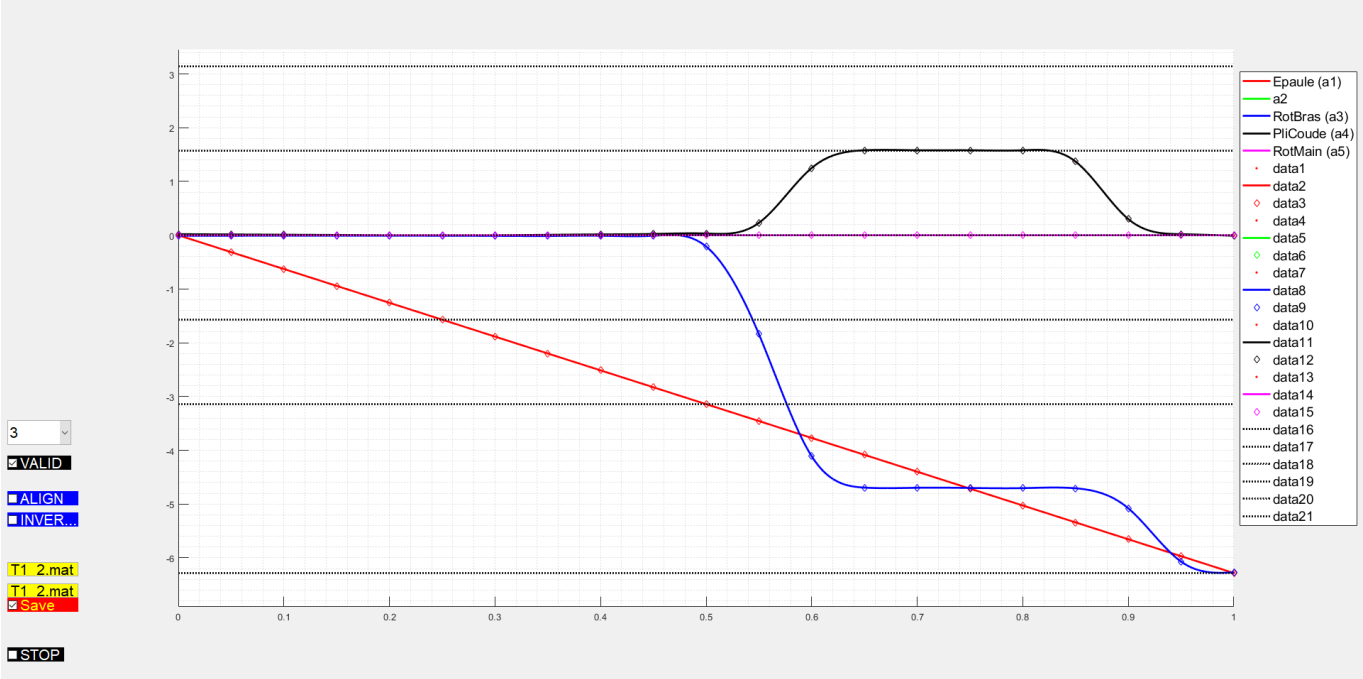


Figure 5: *Trajectoires angulaires de a_1, a_3 et a_4 lors d'un mouvement de crawl*

2.3 Extraction des commandes et choix du type des moteurs

Une fois le mouvement du crawl vérifié et validé sur simulateur, nous avons pu extraire les trajectoires de chacune des articulations et en déduire les moteurs que nous allons utiliser :

- Rotation continue sur a_1 ⇒ utilisation d'un moteur CC avec un encodeur pour asservir en position.
- Enchaînements de rotations de 90° ou 270° sur a_3 ⇒ utilisation d'un moteur CC avec un encodeur pour asservir en position.
- Enchaînements de rotations de 90° dans les 2 sens sur a_4 ⇒ utilisation d'un servo moteur.

Toutes les positions des 3 trajectoires ont été enregistrées dans un fichier CSV afin de pouvoir les utiliser avec l'algorithme de commande.

Trajectoires100V2.csv

```
1     0;          -0.0100;     0.0200
2    -0.0628;     -0.0099;     0.0190
3    -0.1257;     -0.0098;     0.0180
4    -0.1885;     -0.0097;     0.0170
5    -0.2513;     -0.0096;     0.0160
6    -0.3142;     -0.0095;     0.0150
7    -0.3770;     -0.0094;     0.0140
8    -0.4398;     -0.0093;     0.0130
9    -0.5027;     -0.0092;     0.0120
10   -0.5655;     -0.0091;     0.0110
11   -0.6283;     -0.0090;     0.0100
12   -0.6912;     -0.0089;     0.0091
13   -0.7540;     -0.0088;     0.0081
14   -0.8168;     -0.0088;     0.0071
15   -0.8796;     -0.0086;     0.0061
16   -0.9425;     -0.0085;     0.0050
17   -1.0053;     -0.0083;     0.0038
18   -1.0681;     -0.0081;     0.0026
19   -1.1310;     -0.0080;     0.0015
20   -1.1938;     -0.0079;     0.0006
21   -1.2566;     -0.0080;     0.0000
22   -1.3195;     -0.0082;    -0.0003
23   -1.3824;     -0.0086;    -0.0003
24   -1.4452;     -0.0090;    -0.0001
25   -1.5080;     -0.0095;    -0.0000
26   -1.5708;     -0.0100;     0.0000
27   -1.6335;     -0.0104;    -0.0002
28   -1.6963;     -0.0108;    -0.0005
29   -1.7590;     -0.0112;    -0.0007
```

Figure 6: *Début du fichier CSV des positions*

2.4 Réalisation de la maquette

Afin de confirmer notre modèle théorique, nous avons réalisé une maquette du bras. Pour correspondre au modèle simplifié précédemment explicité, celle-ci est constituée de 4 parties :

- Un buste
- Une épaule
- Un bras
- Un avant-bras avec une main

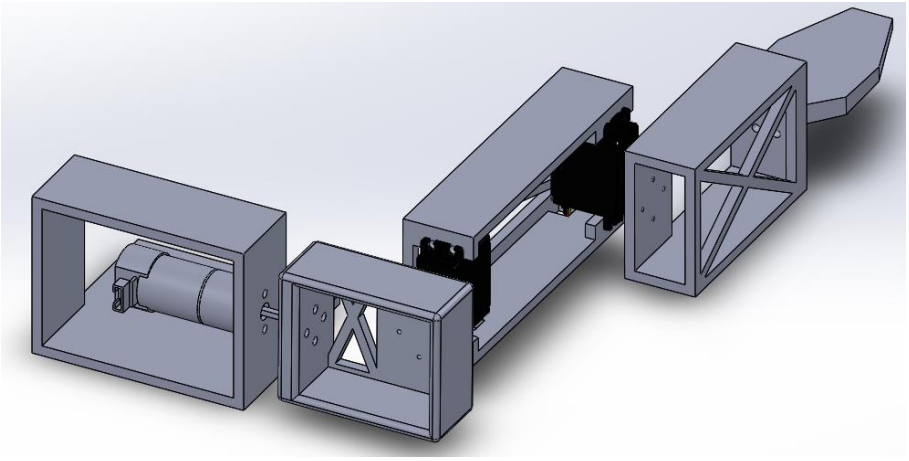


Figure 7: *Maquette du bras – 1ère version*

La *Figure 7* représente la première version de la maquette. Celle-ci est constituée d'un moteur CC avec encodeur et de 2 servos moteurs. Cependant, lors des tests avec celle-ci, nous nous sommes rendu compte que l'articulation a_3 devait pouvoir faire une rotation de 360°. En effet, pour éviter que les fils des moteurs ne s'enroulent autour du bras lors de son mouvement, nous sommes dans l'obligation d'effectuer une

rotation complète sur a_3, mouvement impossible pour un bras humain. Or, un servo moteur est limité à 180°. C'est donc pour cette raison que nous avons choisi d'utiliser un moteur CC à cette position.

Une fois cette contrainte identifiée, les pièces de l'épaule et du bras ont été modifiées pour donner le modèle suivant :

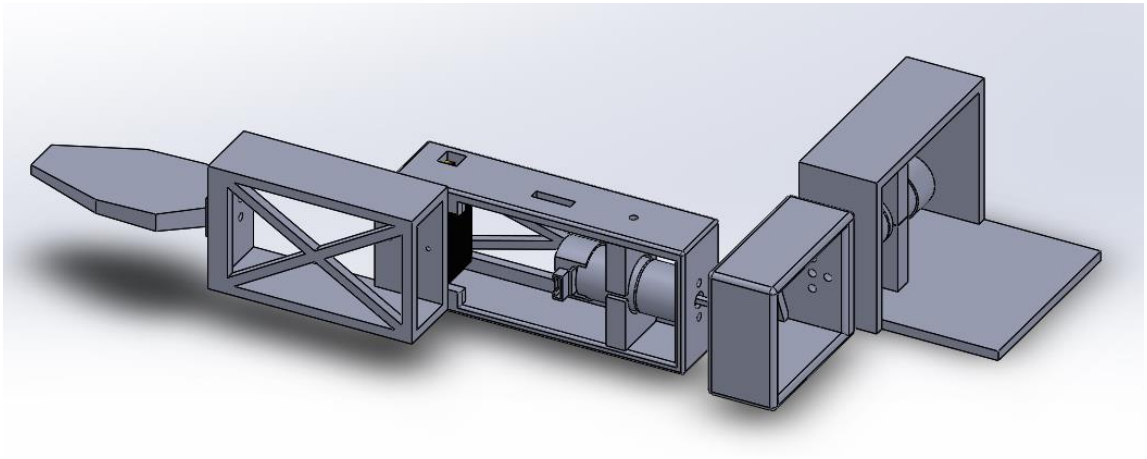


Figure 8: *Maquette du bras – 2ième version*

Etant donné le faible couple que possède notre moteur CC responsable de la rotation de l'épaule (articulation a_1), nous avons été contraints de rajouter un jeu d'engrenages. Ainsi, la liaison entre le buste et l'épaule se fait maintenant grâce à 2 engrenages (80 dents et 16 dents) multipliant le couple par 5.

2.5 Modélisation du robot à l'échelle 1:1

2.5.1 Modélisation du corps

Pour pouvoir embarquer l'électronique nécessaire (*Cf. partie 3.1 & 4*) au bon fonctionnement du robot, nous devons utiliser des tubes de plexiglas hermétiques. Afin de former un corps fonctionnel, il est important de correctement agencer les tubes et de les unir. Pour cela, nous avons opté pour la solution suivante :

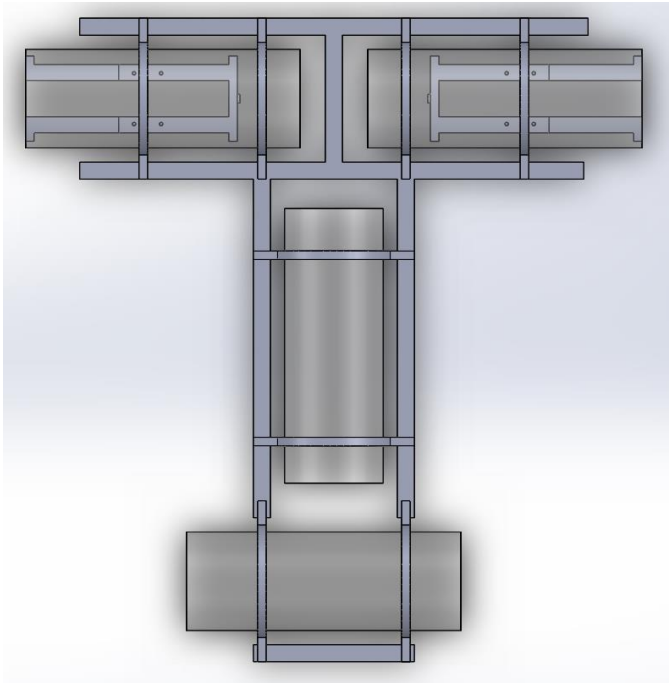


Figure 9: *Corps du robot à l'échelle 1:1*

Comme l'indique la *Figure 9*, les tubes ont été placés de telle sorte à former un buste à la géométrie similaire à celle d'un buste humain :

- 2 tubes pour les épaules
- 1 tube pour le tronc
- 1 tube pour les hanches

Tous les éléments sont reliés par un squelette et y sont attachés grâce à des demi-cercles emprisonnant les tubes et se fixant au squelette.

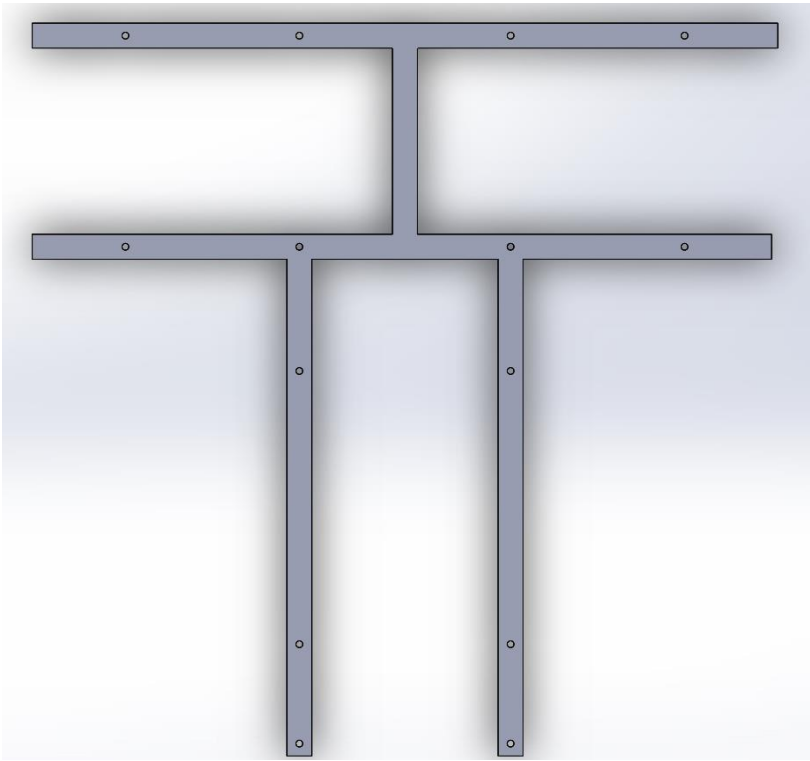


Figure 10: *Squelette du robot*

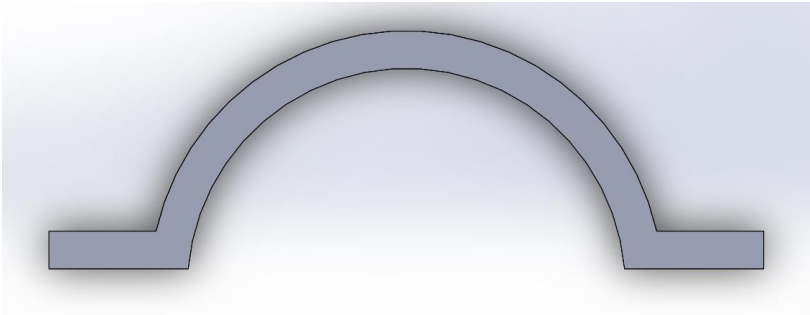


Figure 11: *Attache emprisonnant les tubes*

Le buste du robot fait donc 728mm de large pour 760mm de hauteur. Ces dimensions sont plus importantes que celles d'un buste humain, surtout pour la largeur d'épaules. Cependant, cela augmente sa surface et, par la même occasion, sa flottabilité. S'il s'avère que le robot flotte trop, affectant ainsi son fonctionnement, il suffira de le lester.

Afin de garantir une solidité suffisante, nous avons découpés ces pièces à la CNC, dans des plaques de plexiglas de 1cm d'épaisseur.

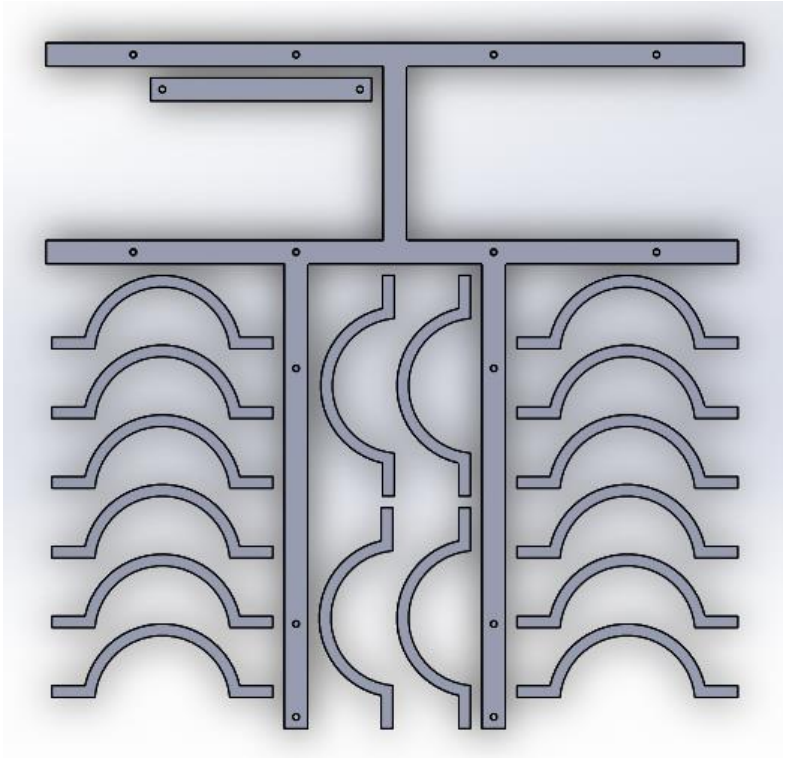


Figure 12: *Disposition des pièces pour une découpe à la CNC*

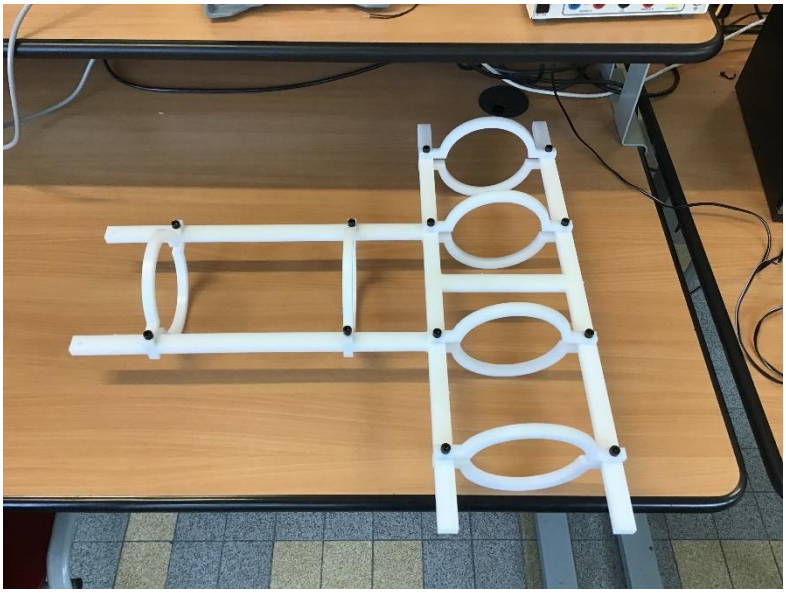


Figure 13: *Aperçu des pièces découpées et assemblées*

2.5.2 Modélisation interne aux tubes

En attendant de recevoir les différents tubes hermétiques, nous avons créé les différents systèmes de fixation des composants à l'intérieur. Les dimensions nécessaires à cette modélisation ont été prise sur un tube issu d'un autre projet.

Pour la fixation des moteurs, nous avons, dans un premier temps, pensé à faire une plaque qui pouvait se fixer de part et d'autre du tube pour garantir sa solidité. En effet, le moteur est lourd et envoie beaucoup de puissance, il fallait donc trouver un système qui permettait de bien le maintenir. Après mure réflexion, nous avons opté pour un système de maintien sur un côté seulement et de l'autre côté, des barres qui viennent s'appuyer contre les parois du tube. La pièce a été conçue afin de faire sortir l'arbre moteur au centre du tube.

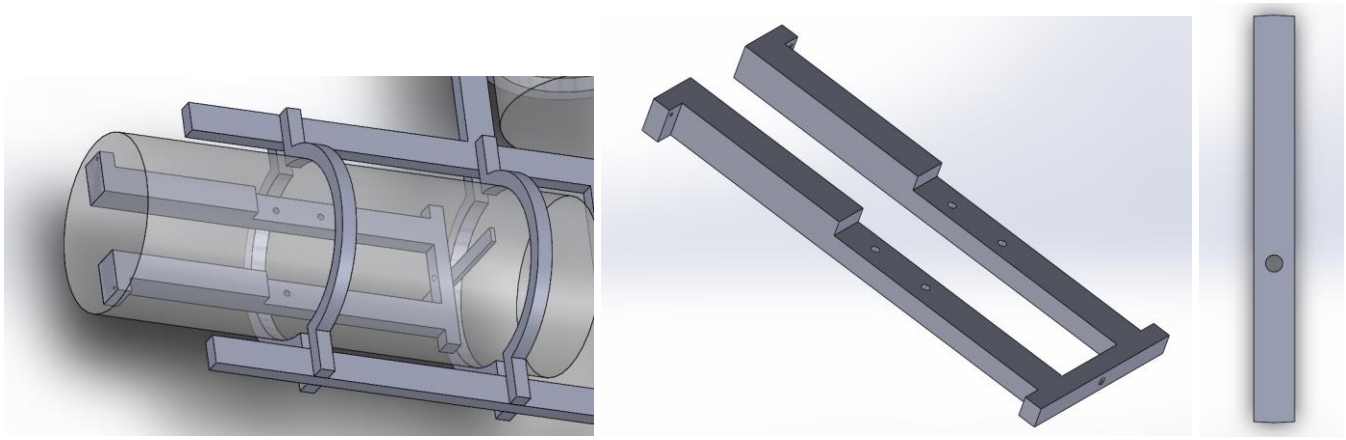


Figure 14: *Attaches internes aux tubes*

Enfin, pour garantir le support de la carte responsable de la commande des moteurs, nous avons modélisé et imprimé en 3D une dernière pièce venant se fixer sur des trous taraudés déjà existants à l'intérieur des tubes.

Figure 15: *Attache de la carte*

2.5.3 Modélisation du bras

Etant donné le poids et le couple à appliquer (*Cf. partie 4.1.1*), nous avons cherché une épaisseur suffisante de matériaux pour assurer la solidité du robot. Pour cela, nous avons choisi une épaisseur de 3cm. Pour des raisons similaires à précédemment, les pièces sont vouées à être découpées à la CNC dans des plaques de 1cm puis assemblées.

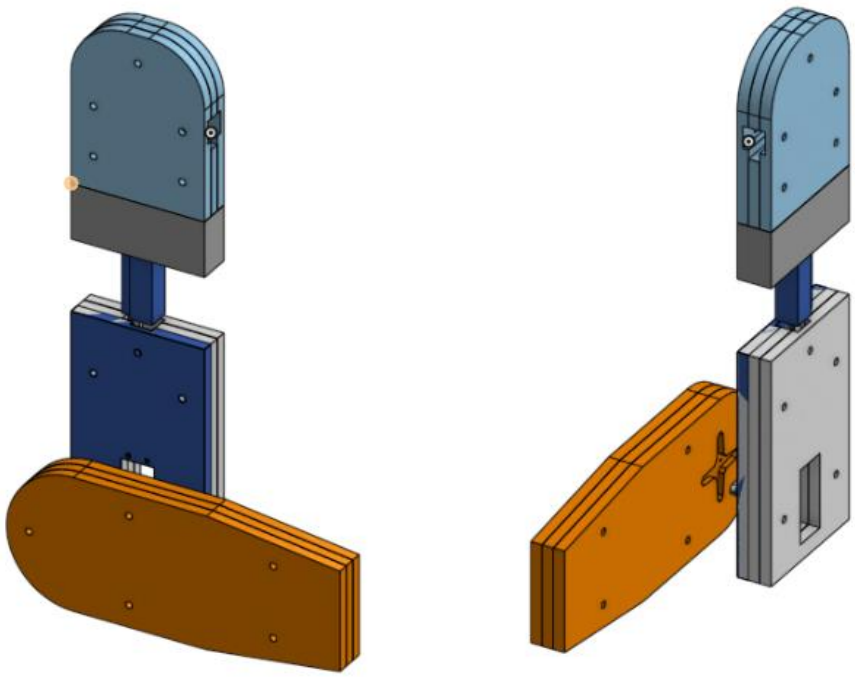


Figure 16: *Modèle du bras assemblé (sans les moteurs)*

Chacune des pièces possède des encoches et emplacements correspondants, respectivement, à l'arrivée d'axes et à l'emplacement de moteurs à intégrer au bras.

3 PROGRAMMATION

3.1 Partie hardware

Concernant l'architecture hardware, nous avons fait dans un premier temps le choix de travailler sur une *Beagle Bone Blue*. En effet, celle-ci dispose d'un panel de ports :

- GPIO
- Connecteurs pour servos moteurs
- Connecteurs pour moteurs CC avec encodeur
- Etc…

Ils nous permettent de travailler directement avec les moteurs que nous avons sélectionnés (*Cf. partie 2.3*) sans qu'il soit nécessaire d'ajouter de composants externes. Nous obtenons donc l'architecture suivante :

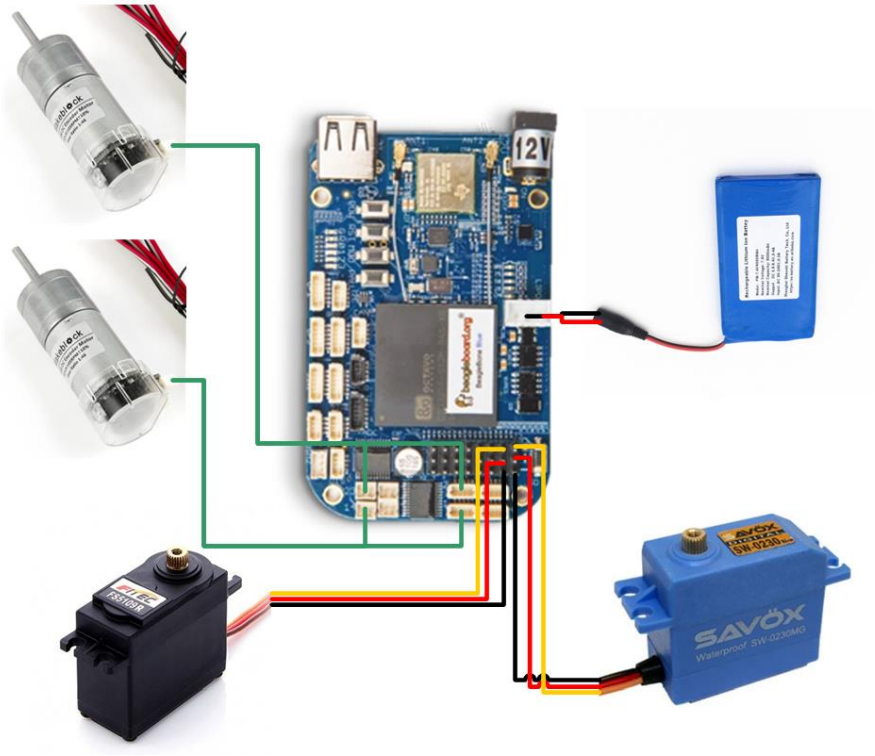


Figure 17: *Architecture hardware de la maquette*

Comme évoqué précédemment, nous utiliserons des moteurs CC avec encodeur pour le bras et pour l'épaule, un servo pour l'avant-bras et un pour la jambe. Nous verrons par la suite qu'il sera nécessaire de modifier ces moteurs pour s'adapter à la structure du robot (moteurs étanches, couples suffisants, etc…). De même, pour la batterie qui devra être changée en conséquence.

Le choix de la *Beagle Bone Blue* a également été fait pour sa possibilité de programmation en wifi. Ainsi, nous pouvons continuer à coder les dernières modifications sans nécessairement ouvrir les tubes hermétiques du système final.

3.2 Partie Software

3.2.1 Environnement de programmation

Afin d'accéder et avoir une meilleure gestion des fichiers sous la *Beagle Bone Blue*, nous travaillons avec *WinSCP*. Concernant la compilation et l'exécution des programmes, nous utilisons *PuTTY* par simplicité. Les *Figure 15* et *Figure 16* permettent donc de comprendre la structure de l'environnement.

Connection Wifi :	
Identifiant	*BeagleBone – XXX*
Mot de passe	*BeagleBone*
Accès via Terminal /Putty/Winscp/… :	
Adresse IP	*192.168.8.1*
Identifiant	*debian*
Mot de passe	*temppwd*

Figure 18: *Identification de la carte*

Sous *PuTTY* ou un terminal :

Identifiant	*Debian*
Mot de passe	*temppwd*
Commandes	*cd rc_project_nageur* *sudo ./rc_project_nageur 1*
Mot de passe	*temppwd*

Figure 19: *Exécution du programme*

Le paramètre '1' passé à l'exécutable permet la bonne exécution du code, nous verrons par la suite son utilité et le rôle d'autres paramètres.

Pour accéder au fichier et modifier le programme, voici les étapes :

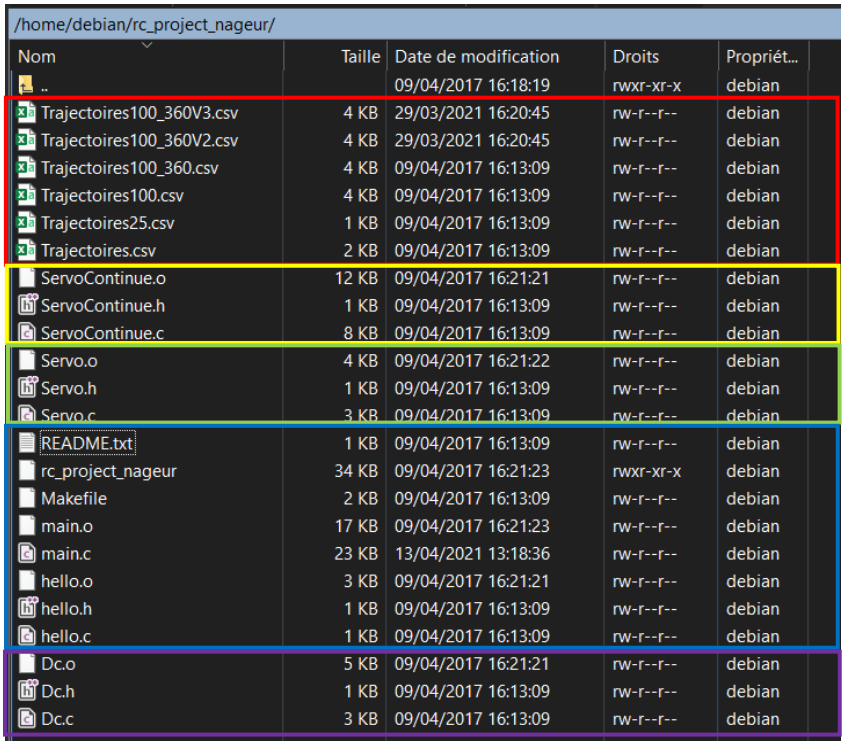
Chemin d'accès au projet	*/home/debian/rc_project_nageur/*	
Vue d'ensemble		
Description		Quelques exemples de mouvements générés dont les trajectoires et le nombre d'échantillons varient
		Fonction de commande d'un servo moteur à rotation continue (pas utilisé dans notre prototype)
		Fonction de commande d'un servo moteur prenant une commande en radians
		Main et autres fichiers
		Fonction de commande d'un moteur DC avec encodeur, prend donc en entrée une vitesse en radians/s

Figure 20: *Étapes pour accéder aux fichiers et les modifier*

3.2.2 Fonction de base

Les fichiers *Dc.c*, *Servo.c* et *ServoContinue.c* permettent de contrôler leurs moteurs respectifs. Voici une brève description de ces derniers :

Fonctions	Description
int **Dc**(double DC1_VITESSE, double DC1_PORT)	Envoie une commande en rad/s au port désiré allant de 1 à 4. Mettre '5' pour envoyer la commande à tous les ports.
int **Servo** (long double S1_ANGLE_RAD, double S1_Hz, double S1_PORT, double S1_DELAY)	Envoie une commande d'angle entre 0 et π radians aux moteurs (S1_ANGLE_RAD), avec une fréquence de commande désirée (S1_Hz conseillée à 50Hz), au port désiré et avec un délai voulu. Le délai peut être utilisé pour rendre la fonction bloquante. Si pas nécessaire, mettre 1.
int **ServoContinue** (long double S1_ANGLE_RAD, double S1_Hz, double S1_PORT, double S1_DELAY)	Même description que précédemment sauf qu'ici S1_ANGLE_RAD est une vitesse angulaire car un servo à courant continu ne se commande pas en position mais en vitesse.

Figure 21: *Description des commandes des moteurs*

Ces fonctions sont inspirées de la librairie fournie sur la *Beagle Bone Blue*. D'autres paramètres peuvent être ajoutés et nécessaires selon les moteurs en question : tension de commande, pulse, etc…, ces derniers étant déjà dans les fichiers mais laissés à des valeurs par défaut.
C'est donc sur la base de ces fonctions que nous allons pouvoir commander nos moteurs et générer notre trajectoire.

3.2.3 Fonctions de contrôle et de génération de trajectoire

Plusieurs fonctions de contrôle des moteurs ont été réalisées. Toutes ne sont pas nécessaires pour l'architecture de commande que nous avons choisie mais pourrait l'être pour une version ultérieure ou l'ont été pour avancer dans notre réflexion.

Fonctions	Description
int **LectureCsv**(void); *[Utilisée]*	Permet de lire les trajectoires accessibles dans un fichier CSV issu de *MATLAB* et de stocker chaque colonne dans un tableau : Colonne 1 ⇒ *traj1* (positions de l'épaule) Colonne 2 ⇒ *traj2* (position du bras) Colonne 3 ⇒ *traj3* (position de l'avant-bras)
int **Dc_Bras**(int DC_PORT, double DC_TEMPS_DEPART, double DC_TEMPS_BOUCLE, int DC_POSITION_DEPART);	Commande du moteur DC du bras en fonction du temps et utilisant les différents plateaux des trajectoires comme paliers à atteindre.
int **Dc_Bras_pos**(int DC_PORT, double DC_POSITION_EPAULE_DEPART, double DC_POSITION_DEPART_BRAS); *[Utilisée]*	Commande du moteur DC du bras en fonction de la position de l'épaule et utilisant les différents plateaux des trajectoires comme paliers à atteindre. Nous faisons ici un asservissement en position du moteur DC du bras pour vérifier sa précision.
int **Dc_Trapeze**(int DC_PORT, double DC_POSITION_EPAULE_DEPART, double	Commande en trapèze 1/3, 1/3, 1/3 en fonction du temps.

DC_TEMPS_DEPART, double DC_TEMPS_BOUCLE, double DC_POSITION_EPAULE_DESIRE);	
int **Dc_Bras_Trapeze**(int DC_PORT, double DC_POSITION_EPAULE_DEPART, double DC_POSITION_DEPART_BRAS, double COMPTEUR_TOUR); *[Utilisée]*	Séquence de commande en trapèze du bras pour générer le mouvement final. Chaque commande interne est faite avec un trapèze en position.
nt **Dc_Trapeze_pos**(int DC_PORT, double DC_POSITION_DEPART, double DC_POSITION_DESIRE, double DC_VITESSE_PLATEAU); *[Utilisée]*	Commande en trapèze en position avec asservissement.
int **Servo_Jambe**(int SERVO_PORT_JAMBE, int DC_EPAULE_PORT, int DC_POSITION_DEPART_EPAULE); *[Utilisée]*	Séquence de battements de jambe cadencés sur la rotation du bras : 2 battements par rotation.
int **Initialisation**(void); *[Utilisée]*	Initialisation des moteurs pour envoi de commande.
int **End**(void); *[Utilisée]*	Boucle infinie de fin de programme.

Figure 22: *Fonctions de contrôle et de générations de trajectoire*

A partir de quelques-unes de ces fonctions, nous allons donc pouvoir commander le mouvement du bras complet.

3.3 Algorithmie et implémentation

3.3.1 Choix de l'algorithme

Le robot nageur est un robot se déplaçant dans un environnement incertain. En effet, plusieurs erreurs peuvent être dues à son implémentation en mer : vagues, vent, etc… En conséquence, nous avons fait le choix d'exclure tout algorithme basé sur le temps. Ainsi, les commandes en trapèze en temps sont remplacées par des commandes en trapèze en position. De ce fait, il n'y a pas une accélération pendant 1/3 du temps total de la commande mais pendant 1/3 de la trajectoire. De cette manière, nous pourrons réaliser un asservissement en position de chaque moteur plus aisément.

De plus, pour sécuriser davantage le fonctionnement de ce système, nous avons fait le choix de synchroniser nos moteurs pour qu'il n'y ait pas de déformation du mouvement du bras. En effet, le bras étant constitué de 3 moteurs soumis à des forces différentes, chaque moteur peut donc avoir un certain retard (dû à ces contraintes) ou même une certaine avance (accélération due au mouvement circulaire). Nous remarquons rapidement que le mouvement du bras est cadencé par la vitesse de rotation de l'épaule. Ce phénomène de dépendance se retrouve notamment sur les autres moteurs, formant la hiérarchie de contrôle suivante :

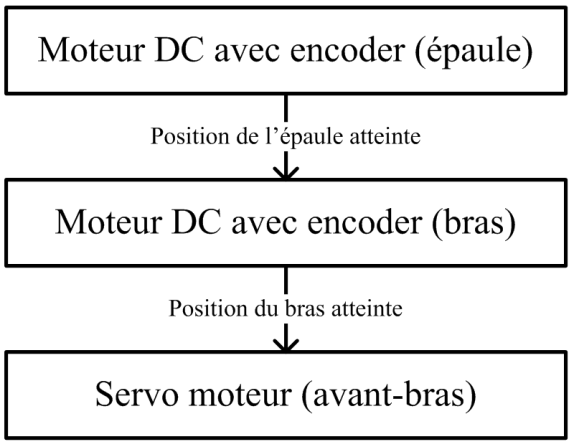


Figure 23: *Hiérarchie d'asservissement*

C'est sur cette hiérarchie que se base notre algorithme. A cela s'ajoute l'étude préalable faite sur *MATLAB* générant les positions de chaque moteur. Par analyse de ces trajectoires, on en déduit un découpage de la commande.

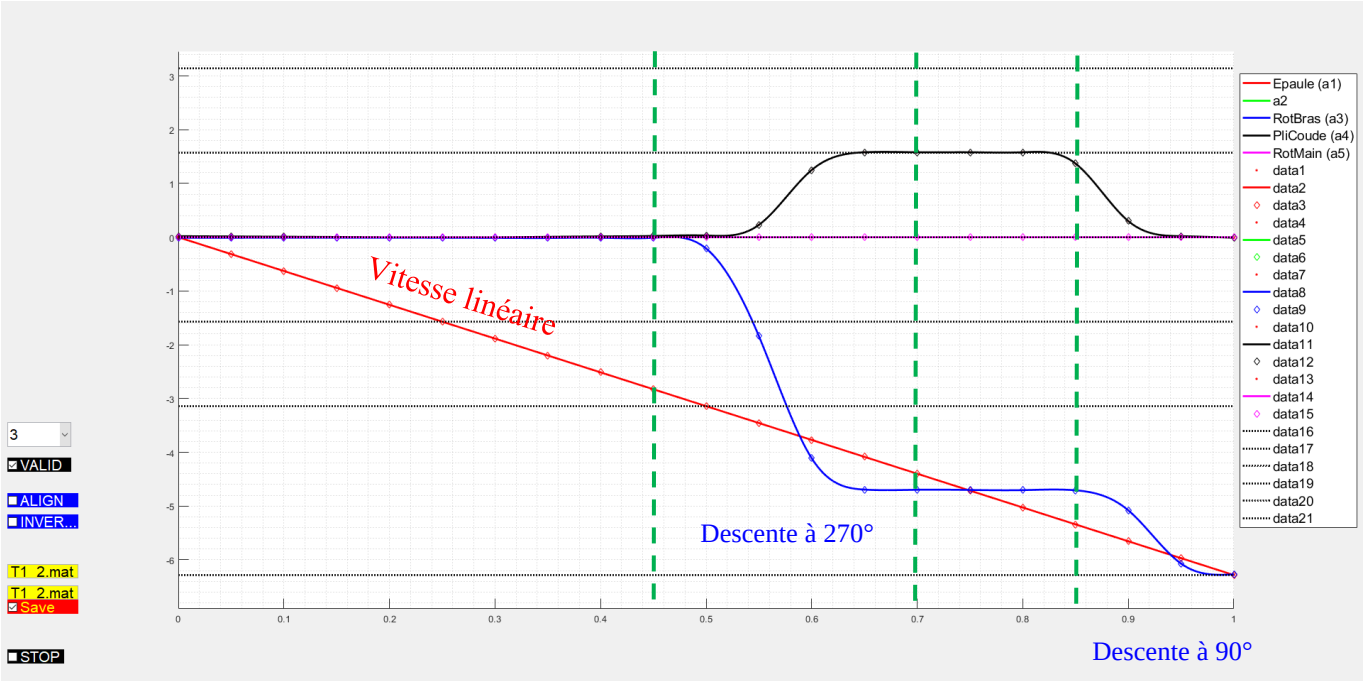


Figure 24: *Découpage de la commande*

De ce découpage en découle les étapes de commandes suivantes :

Epaule	Commande en vitesse angulaire constante
Bras	Conditions initiales : $\theta_{bras} = 0, \theta_{épaule} = 0, \theta_{avant-bras} = 0$ Si $\theta_{épaule} < 0.45 * 2\pi$ alors $V_{bras} = 0$ Sinon, si $\theta_{épaule} < 0.70 * 2\pi$ alors $V_{bras} = V_1$ Sinon, si $\theta_{épaule} < 0.85 * 2\pi$ alors $V_{bras} = 0$ Sinon $V_{bras} = V_2$
Avant-bras	Commande par lecture du fichier CSV contenant les trajectoires. Puisque le fichier comprend 100 échantillons par articulation, la position de l'avant-

	bras sera actualisée tous les 1% du mouvement global. Une telle précision n'est pas forcément nécessaire.

Figure 25: *Étapes de commande*

3.3.2 Implémentation

Pour contrôler le mouvement du bras nous pouvons jouer sur quelques paramètres de commande.

```
27  //COMMANDE
28  #define nb_commande             100
29  #define nb_echantillon_voulu    20
30  #define temps_boucle            3.50000000000
31  #define rapport_engrenage       5
32  #define tour_bras               1500
33
34  //ASSERVISSEMENT
35  #define coef_vitesse_Bras       1
36  #define coef_position_Bras      1
37  #define coef_vitesse_Epaule     1
38
39  //PORTS
40  #define PORT_SERVO_AVANT_BRAS_G 1
41  #define PORT_SERVO_JAMBE_G      2
42  #define PORT_DC_BRAS_G          1
43  #define PORT_DC_EPAULE_G        2
```

Figure 26: *Paramètres de commande*

- *nb_commande* : le nombre de lignes du fichier CSV sélectionné.
- *nb_echantillon_voulu* : sert à augmenter ou diminuer la précision de l'avant-bras. Ici, entre 10 et 20 échantillons suffisent.
- *temps_boucle* : il s'agit du temps que met l'épaule pour faire une rotation complète. Ainsi, nous jouons sur la vitesse de nage. Il faut cependant souligner que si le temps est trop rapide, le mouvement ne sera pas esthétique et bien formé. En effet, les moteurs n'auront pas tous le temps de bien faire leurs commandes en trapèze. Par expérimentation, on notera un temps minimal de 1.5s conseillé avec ce prototype. A l'opposé, si le mouvement est trop long, le moteur peut parfois manquer de couple et donc avoir du mal à finir sa boucle.
- *coef_vitesse_Bras*, *coef_position_Bras*, *coef_vitesse_Epaule* : nous avons également rajouté des coefficients s'il est nécessaire d'ajuster la vitesse de certains moteurs. Dans notre cas, comme il n'y a pas de perturbation extérieure, il n'est pas nécessaire de les modifier.

Concernant le code, nous obtenons donc l'architecture de commande suivante *:*

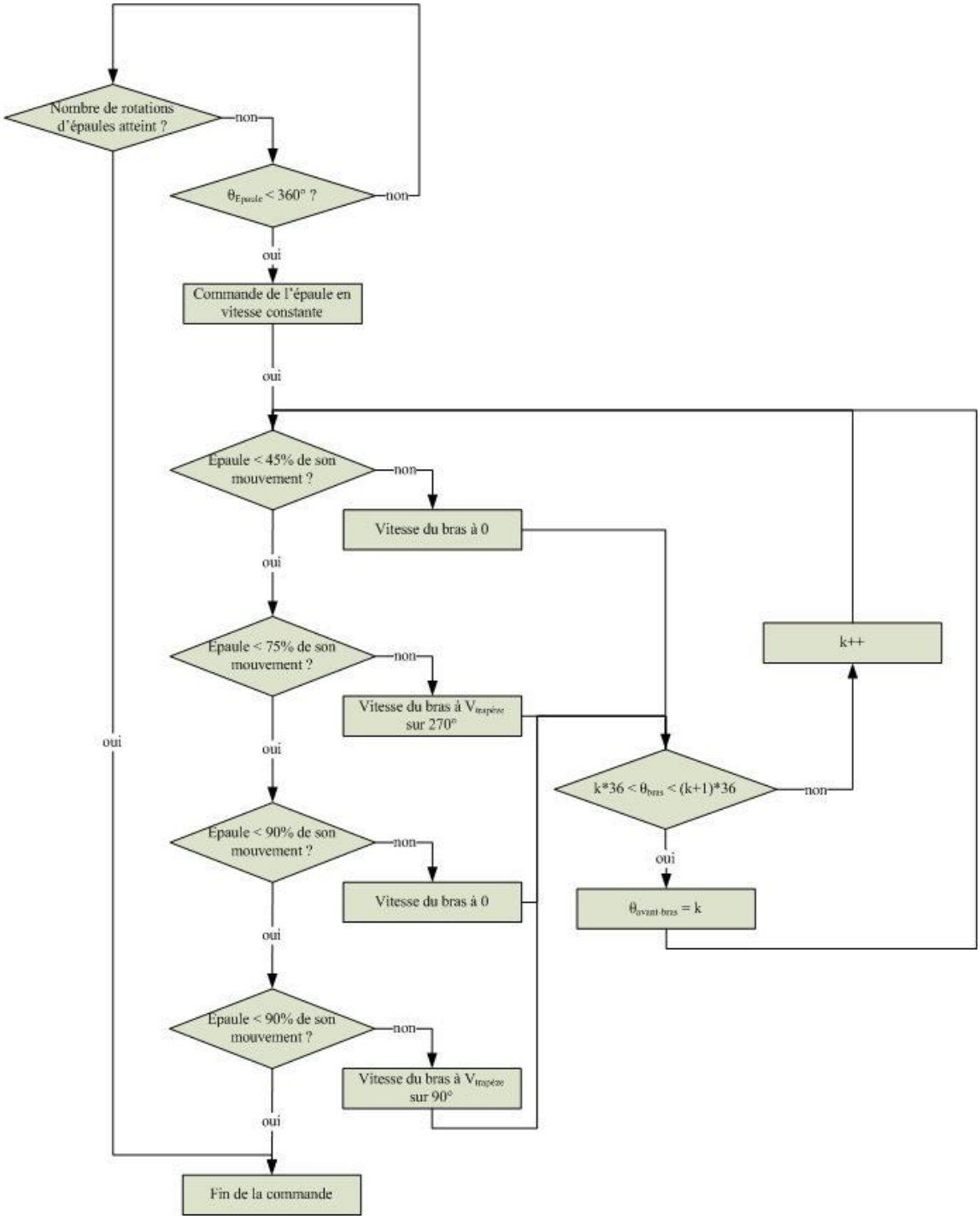


Figure 27: *Architecture de commande*

En définitive, il est donc possible de commander le mouvement de crawl d'un robot nageur avec la méthode proposée. Par ses asservissements successifs, elle garantit la préservation du mouvement d'un point de vue fonctionnel et visuel. Voici le rendu de la méthode proposée :

https://youtu.be/kqU7eqa2u2k

Figure 28: *Lien vers une démonstration*

4 IMPLEMENTATION

4.1 Tests du moteur pour l'épaule

En parallèle des 2 parties précédentes, nous avons cherché quels moteurs utiliser sur le modèle à l'échelle 1:1. Étant donné le type de robot à créer (forme, poids et taille proches de ceux d'un humain) et l'environnement dans lequel celui-ci va évoluer (eau de mer), il nous faut disposer de moteurs assez puissants et possédant un couple suffisant pour assurer le bon déplacement du robot. Pour cela nous avons donc étudié et testé divers moteurs.

Dans un premier temps, après avoir démonté un moteur brushless d'une patte du *REX*, nous avons cherché à connaître le couple qu'il peut déployer. Nous avons effectué différents tests et avons développé un système de maintien du moteur. Pour cela, nous avons utilisé une carte *Beagle Bone Blue*.

Un moteur brushless est un moteur composé de bobines qui sont alimentées de façon séquentielle, c'est-à-dire que le champ magnétique créé entre les bobines se déplace à la même fréquence que la tension d'entrée. Ainsi il faut donc modifier constamment la tension d'entrée afin de s'assurer que le champ soit en avance sur la position du rotor qui cherche à s'orienter dans le même sens que le champ, et donc qui crée le couple moteur. C'est pour cette raison que nous utilisons un *ESC* (*Basic ESC BlueRobotics 2-6S/ 7-26V*).

Un *ESC* (*Electronic Speed Controller*) est un contrôleur spécifique aux moteurs brushless puisqu'il envoie un signal plus ou moins fort au moteur et répond donc au principe présenté précédemment.

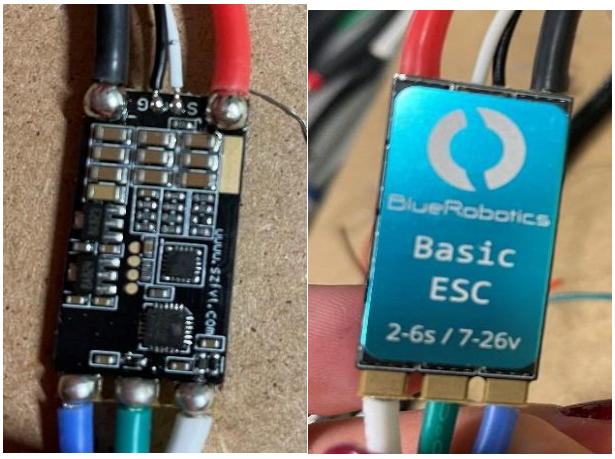


Figure 29: *Photos de l'ESC utilisé*

4.1.1 Tests du couple moteur

Le système d'attache du moteur consiste en un coude d'acier auquel nous avons vissé le moteur et ajouté un serre-câble maintenant le moteur afin qu'il ne se déplace pas avec la puissance et la force qu'il va appliquer. Nous avons ensuite modélisé et créé, à la CNC, une barre de 30cm disposant de 5 trous, permettant l'attache de poids, à laquelle nous avons fixé un marteau en guise de poids.

Ce système nous a permis de vérifier si le moteur était capable de soulever le marteau sans difficulté ou non. Le test étant concluant, le moteur a tourné correctement. Nous avons donc ensuite modélisé et imprimé en 3D une pièce supplémentaire permettant d'ajouter un encodeur sur le moteur, afin de nous donner la position angulaire du moteur.

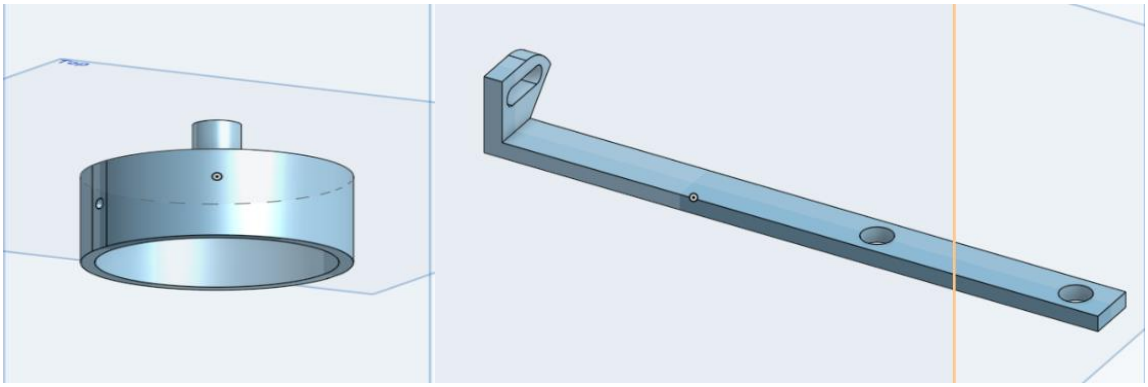


Figure 30: *Modélisation des attaches du moteur*

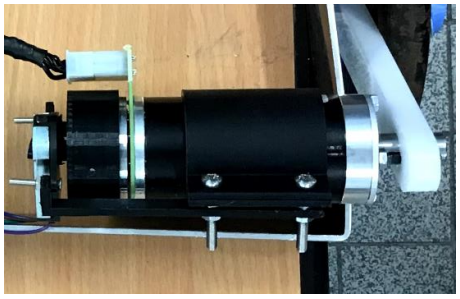
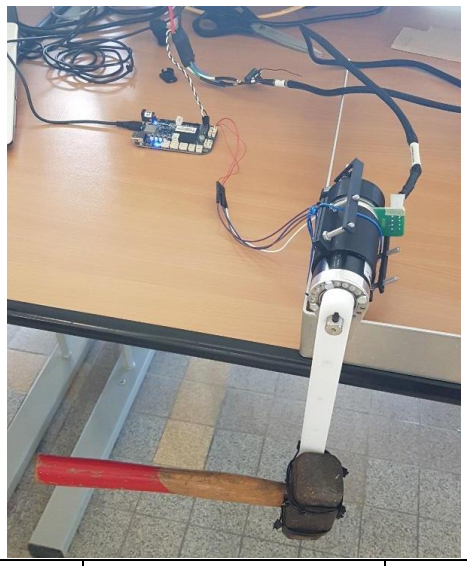


Figure 31: *Attaches du moteur en place*

L'ajout de l'encodeur est nécessaire pour vérifier le maintien de la position donnée malgré la présence du poids. Pour calculer le couple déployé par le moteur, nous avons observé l'évolution du courant traversant le moteur.



I (A)	Poids (kg)	Distance poids – arbre moteur (cm)
1.6	2	15.5

Figure 32: *Photo et conditions des tests*

Calcul du couple : $C = d * F = d * mg = 15.5 * 10^{-2} * 2 * 10^{3} * 9.81 = 3.04\ Nm$

D'après les tests précédents, pour soulever un marteau de 2kg, le moteur nécessite 1.6A sur les 3 possibles. Il déploie ainsi un couple de 3.04Nm, ou une force de 19.62N. Selon certaines études (*Cf partie 6.3*), un

nageur déploie une force de poussée d'environ 5N dans une piscine. Due à la présence de courants et de vagues, nous pouvons aisément affirmer qu'il devra déployer une plus grande force. Néanmoins, avec un courant de 1.6A, nous déployons un couple qui semble suffisant pour nager en sachant que nous pouvons faire plus avec ce même moteur. Afin de confirmer ces hypothèses, nous avons procédé à une nouvelle batterie de tests, cette fois-ci avec un poids de 2.5kg.

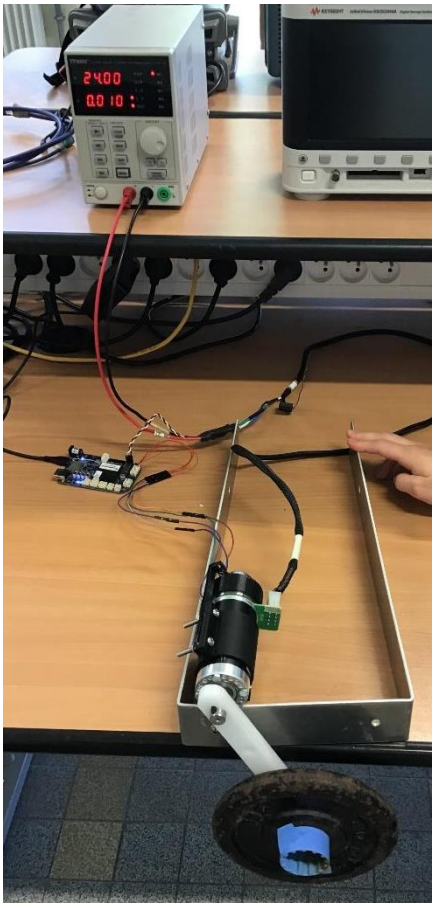


Figure 33: *Tests avec 2.5kg*

4.2 Recherche de composants

Après ces différents tests, nous avons fait une recherche de composants pour la création du robot à l'échelle 1:1, notamment sur les moteurs à utiliser. Nous avons réussi à retrouver le même moteur ainsi que le réducteur sur lesquels nous avons fait des tests. Nous avons également cherché des joints spi et des coupleurs d'axes. En effet, ils nous sont nécessaire afin de faire sortir l'axe moteur des tubes étanches. Notre arbre moteur ayant un diamètre de 12mm, nous avons réfléchi à un moyen de le faire rentrer dans le coupleur d'axe qui a un diamètre de maximum 8mm. N'ayant pas reçu les coupleurs avant la fin des séances de projets, nous n'avons pas pu tester le réducteur.

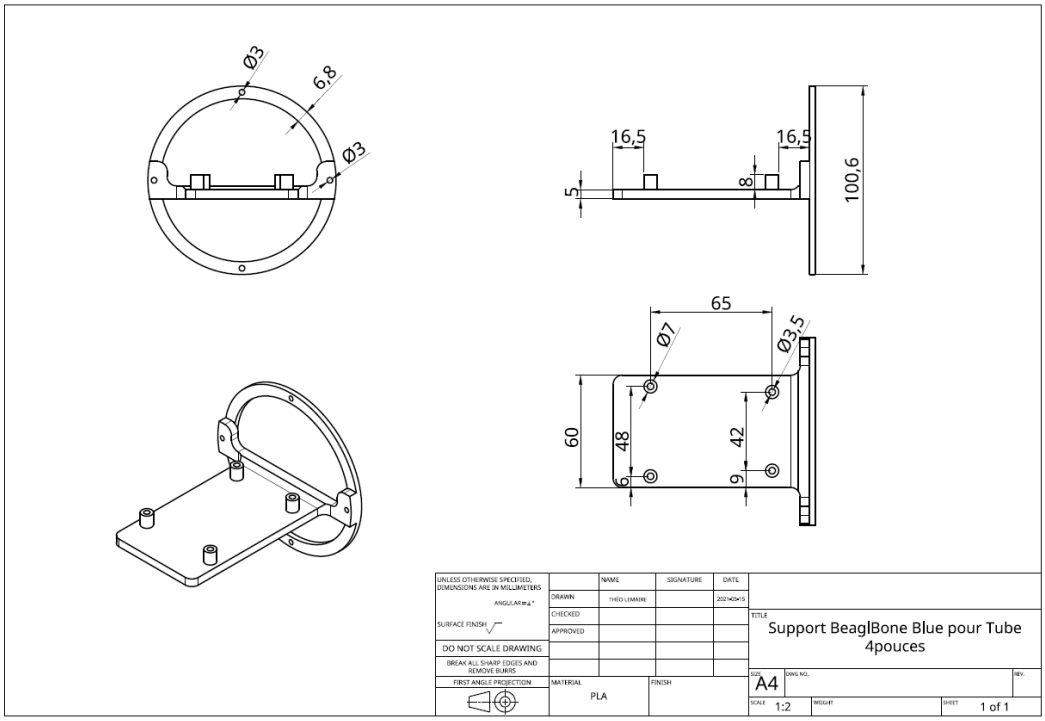
5 SURETE DE FONCTIONNEMENT

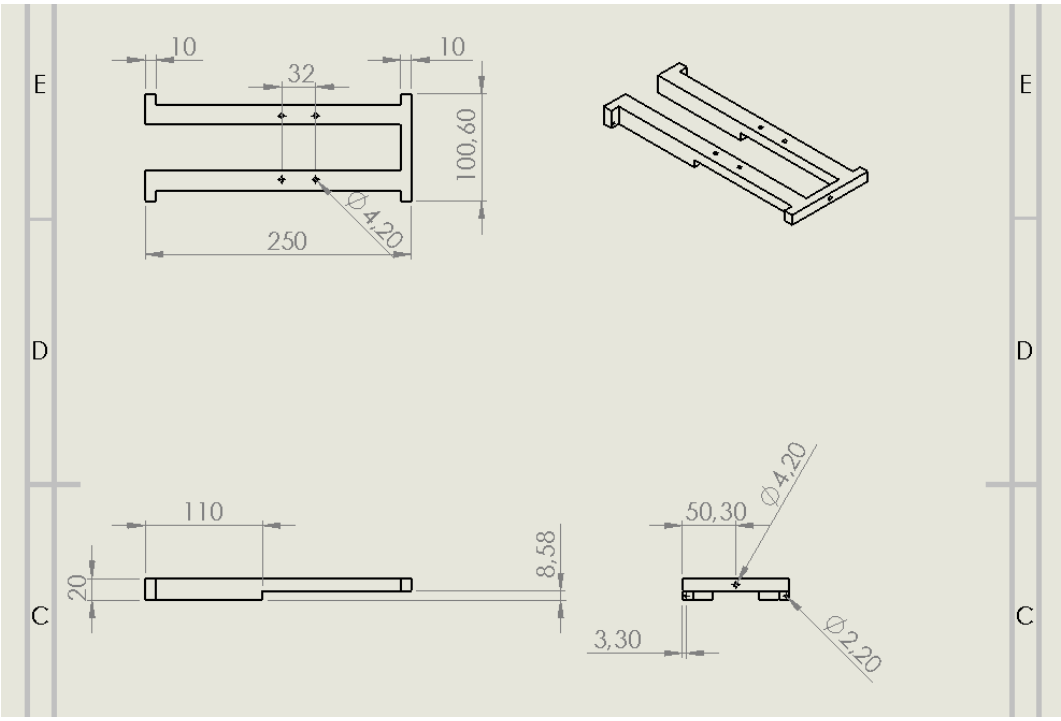
Problèmes possibles	Causes possibles	Résolutions à envisager
Perte du suivi de bouée (or du champ de vision)	Ensoleillement provoque des reflets endommageant l'algorithme de détection	Si courte durée alors garde le cap
	Une vague submerge la caméra	

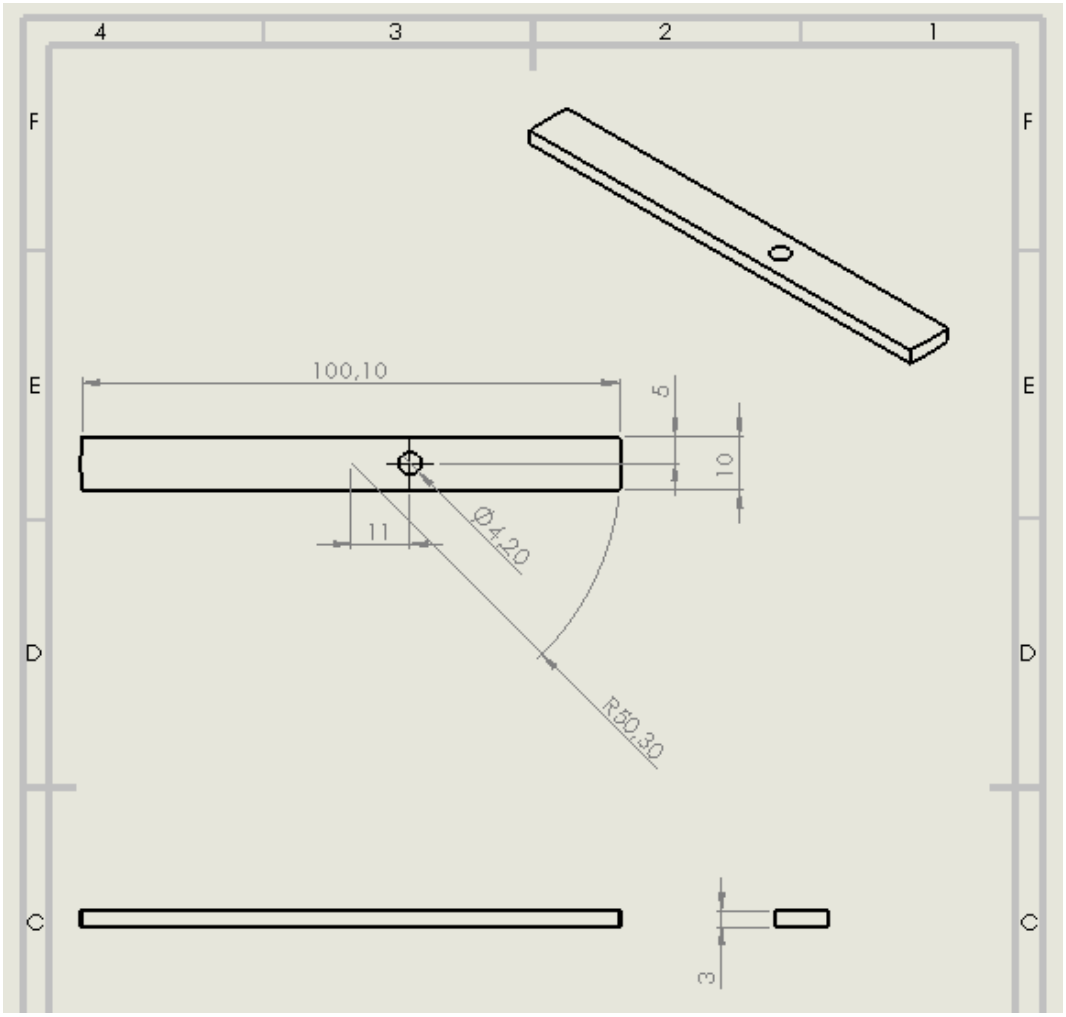
	Caméra HS ou s'est décrochée	Si persiste alors il faut tout stopper
	Carte 1 HS	
Moteur(s) HS	Force appliquée pour le mouvement trop importante	Changement du moteur, et ré-étude du choix moteur
	Défaillance	Si un des moteurs des jambes est HS → continuation de la mission seulement avec les bras (arrêt de la jambe opérationnelle)
		Si un des moteurs des bras est HS → arrêt total du robot.
	Contact avec l'eau (pour les non-étanches)	Changement de certaines parties du robot si manque d'étanchéité
Casse d'une articulation	Résistance trop faible des matériaux	Nouvelle modélisation de la pièce suivant la cause de la casse (épaisseur, taille…)
	Force trop forte appliquée sur l'eau	
	Choc avec divers objets (planche de surf, bouée, cailloux…)	Refabrication de l'articulation (avec un matériau différent suivant la cause de la casse)
	Dégradation due à l'eau salée (rouille, détérioration)	
Carte HS	Court-circuit (prise d'eau)	Changer la carte
	Batterie HS	Refaire l'étanchéité du tube
	Détachée dans le tube et a reçue des chocs	Changer l'attache de la carte par un modèle plus solide
	Surchauffe	Modifier la programmation ou la disposition dans le tube
Batteries tête/corps HS	Batterie épuisée	Changer la batterie (redimensionnement si nécessaire)
	Contact avec l'eau	Refaire/vérifier l'étanchéité
Déviation de trajectoire	Un bras plus rapide	Réadaptation des moteurs du point de vue du code (ne pas mettre exactement les mêmes vitesses)
	Vague	
	Mauvais calcul de trajectoire	Ajout de la prise en compte d'une erreur dans le code
	Mauvaise interprétation des données (erreur vue comme trajectoire, moteur pas précis, mauvaise transmission et modifie la valeur…)	
	Défaillance des moteurs d'un bras	Vérification des valeurs reçues avec des encodeurs
Coule ou flotte	Répartition poids/volume	Mesures/calculs pour meilleur répartition
	Salinité de l'environnement ou densité	Mesures de l'environnement pour adaptation

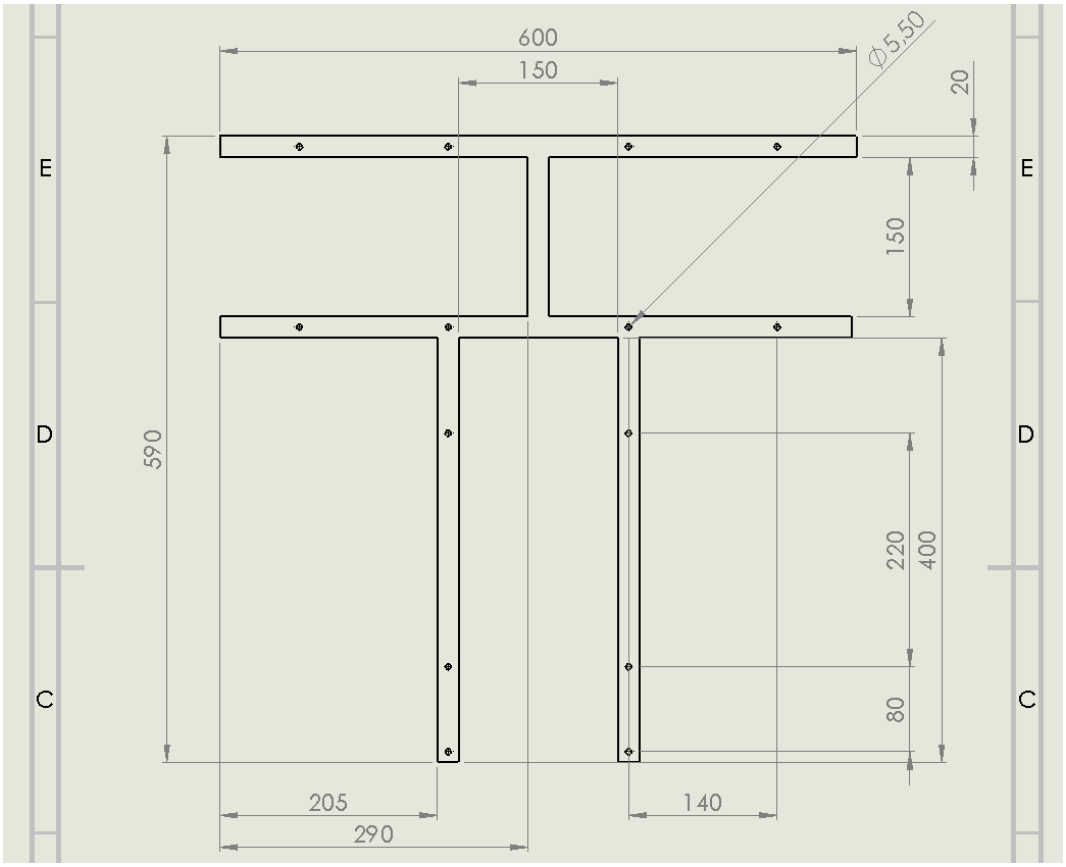
6 ANNEXE

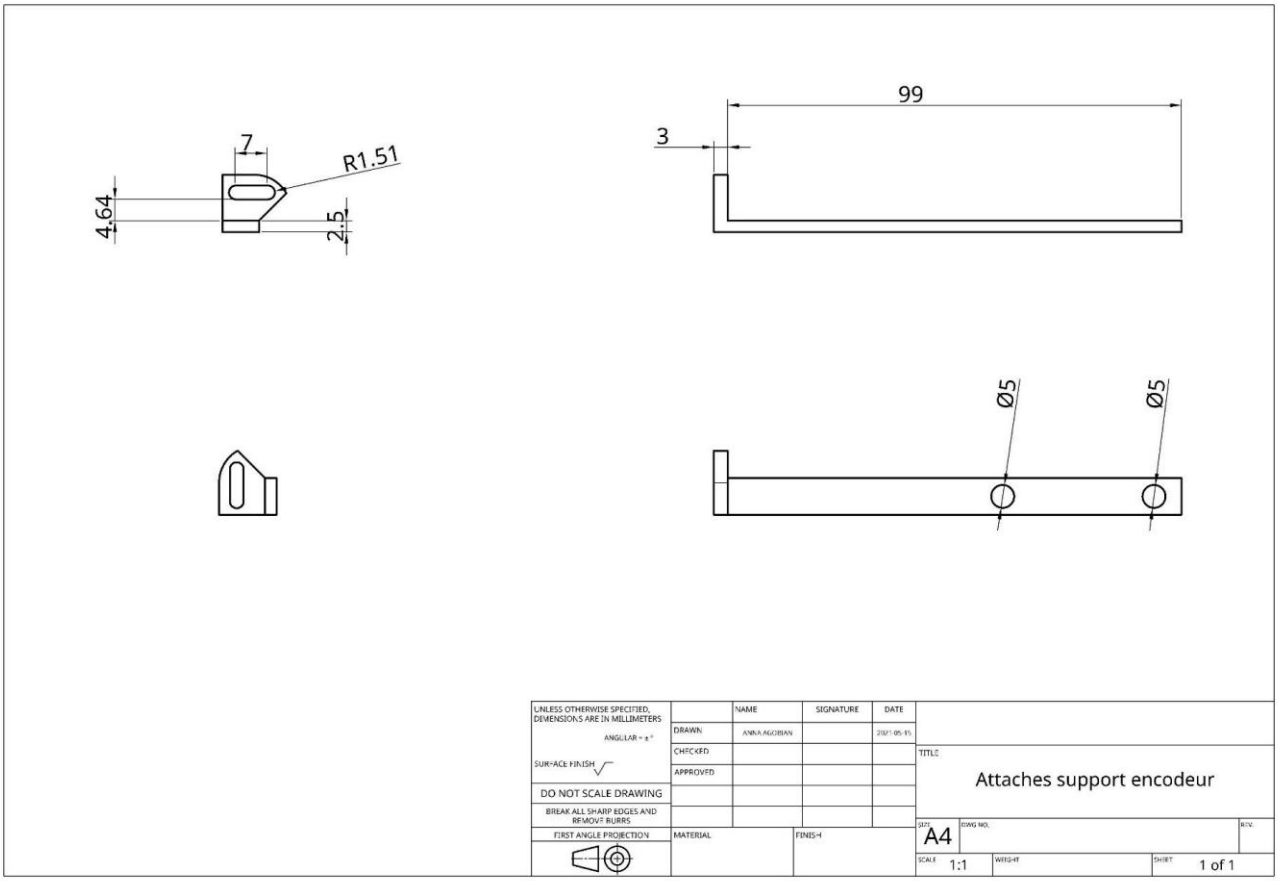
6.1 Modélisation

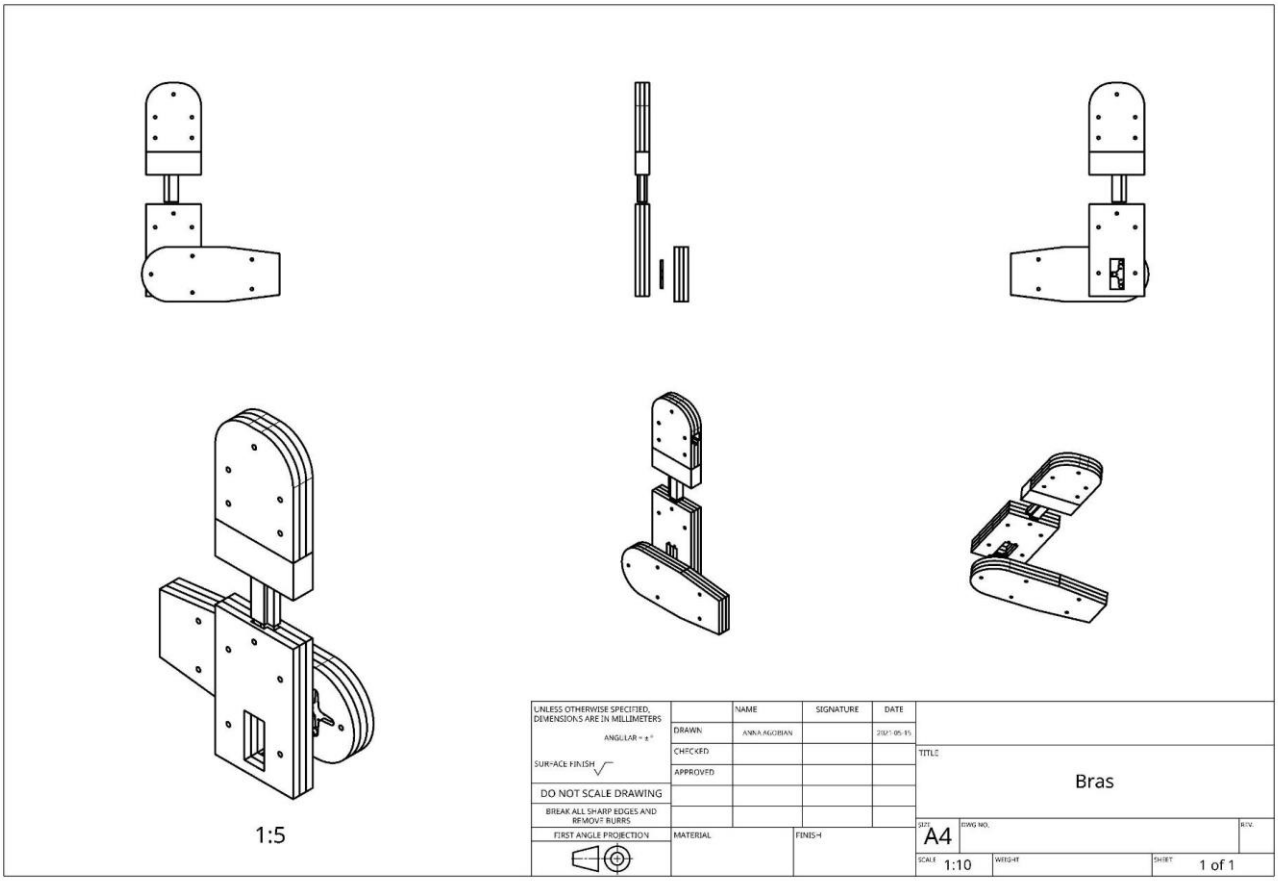


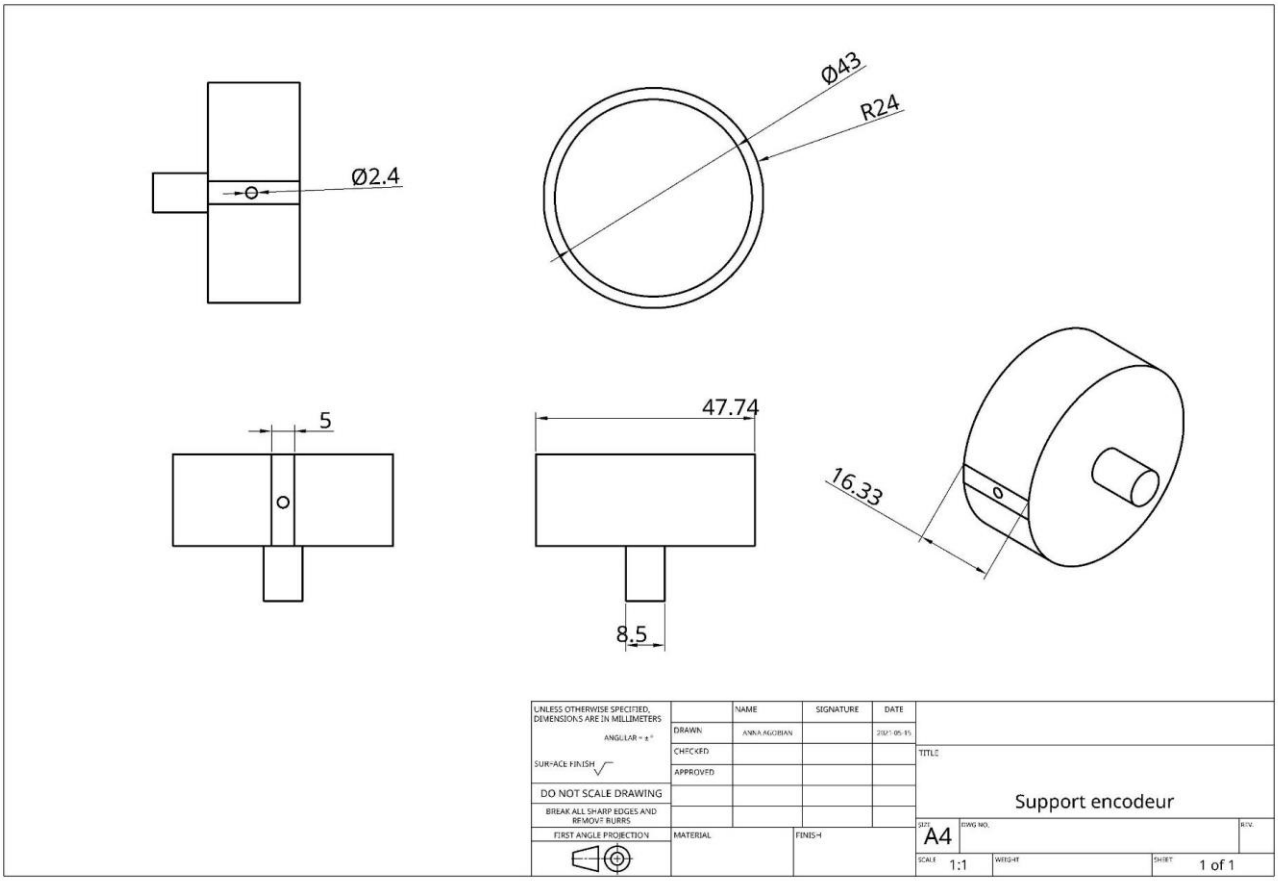



4
3
2
1
F
E
D
C
100,10
5
10
11
Ø4,20
R50,30
3


600
150
Ø5,50
20
150
590
220
400
80
205
290
140


7
R1.51
4.64
2.5
99
3
Ø5
Ø5
UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN MILLIMETERS
ANGULAR = ± °
SURFACE FINISH
DO NOT SCALE DRAWING
BREAK ALL SHARP EDGES AND REMOVE BURRS
FIRST ANGLE PROJECTION
NAME
SIGNATURE
DATE
DRAWN
ANNA AGOBIAN
CHECKED
APPROVED
MATERIAL
FINISH
TITLE
Attaches support encodeur
SIZE
A4
DWG NO.
REV.
SCALE
1:1
WEIGHT
SHEET
1 of 1


1:5
UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN MILLIMETERS
ANGULAR = ± °
SURFACE FINISH
DO NOT SCALE DRAWING
BREAK ALL SHARP EDGES AND REMOVE BURRS
FIRST ANGLE PROJECTION
NAME
SIGNATURE
DATE
DRAWN
ANNA AGOBIAN
CHECKED
APPROVED
MATERIAL
FINISH
TITLE
Bras
SIZE
A4
DWG NO.
REV.
SCALE
1:10
WEIGHT
SHEET
1 of 1



6.2 Programmation

```
//LANCEMENT D'UNE BOUCLE
while (rc_get_encoder_pos(PORT_DC_EPAULE_G) < rapport_engrenage*tour_bras*(compteur_boucle_total+1)
    && rc_get_state() != EXITING)
{
    //GESTION DE L'EPAULE
    Dc(VITESSE_DC_EPAULE_G,PORT_DC_EPAULE_G);

    //GESTION DU BRAS
    Dc_Bras_Trapeze(PORT_DC_BRAS_G, position_depart_epaule, position_depart_bras, compteur_boucle_total);

    //GESTION DE L'AVANT BRAS A PARTIR DES SPLINS
    if(rc_get_encoder_pos(PORT_DC_EPAULE_G)/rapport_engrenage > tour_bras/nb_echantillon_voulu*nb_increment)
    {
        //printf("compteur++");
        compteur_echantillon=compteur_echantillon + increment_echantillon;
        printf("traj3: %lf\n",traj3[compteur_echantillon]);
        nb_increment = nb_increment + 1;
    }
    if(compteur_echantillon > nb_commande ) { compteur_echantillon  = nb_commande;}
    Servo(traj3[compteur_echantillon]-0.5,50,PORT_SERVO_AVANT_BRAS_G,1);

    //GESTION JAMBE
    Servo_Jambe(PORT_SERVO_JAMBE_G , PORT_DC_EPAULE_G, position_depart_epaule);

}
```

```c
//GESTION BRAS - TRAPEZE AVEC ASSERVISSEMENT EN POSITION
int Dc_Bras_Trapeze(int DC_PORT, double DC_POSITION_EPAULE_DEPART, double DC_POSITION_DEPART_BRAS, double COMPTEUR_TOUR)
{
    double position_depart_trapeze;

    //PLATEAU 1 - < 45%
    if((rc_get_encoder_pos(PORT_DC_EPAULE_G)/rapport_engrenage - DC_POSITION_EPAULE_DEPART)< tour_bras*0.45)
    {
        if(antiWhile==0)
        {
            printf("->P1_DC_BRAS\n");
            Dc(0,DC_PORT);
            antiWhile = 1;
        }
    }

    //MONTE A 180+90deg
    else if ((rc_get_encoder_pos(DC_PORT)-DC_POSITION_DEPART_BRAS)<(tour_bras*0.75))
    {
        if(antiWhile==1)
        {
            printf("Position Epaule:   %d\n",(rc_get_encoder_pos(PORT_DC_EPAULE_G)/rapport_engrenage));
            printf("Position Bras:     %d\n",rc_get_encoder_pos(DC_PORT));
            position_depart_trapeze = rc_get_encoder_pos(DC_PORT);
            printf("->270\n");
            antiWhile = 2;
        }
        Dc_Trapeze_pos(DC_PORT, position_depart_trapeze,3.1415*2*0.75+(COMPTEUR_TOUR*2*3.1415), 3.1415*11*2/(temps_boucle));

    }
    //PLATEAU 2
    else if((rc_get_encoder_pos(PORT_DC_EPAULE_G)/rapport_engrenage -DC_POSITION_EPAULE_DEPART)>= tour_bras*0.45
        && (rc_get_encoder_pos(PORT_DC_EPAULE_G)/rapport_engrenage -DC_POSITION_EPAULE_DEPART)<  tour_bras*0.90)
    {
        if(antiWhile==2)
        {
            printf("Position Bras:    %d\n",rc_get_encoder_pos(DC_PORT));
            printf("->P2\n");
            Dc(0,DC_PORT);
            antiWhile = 3;
        }
    }
     //MONTE A 360deg
     else if ((rc_get_encoder_pos(DC_PORT)-DC_POSITION_DEPART_BRAS)<(tour_bras))
     {
         if(antiWhile==3)
         {
             printf("position encodeur: %d\n",rc_get_encoder_pos(PORT_DC_EPAULE_G));
             position_depart_trapeze = rc_get_encoder_pos(DC_PORT);
             printf("->360\n");
             antiWhile = 4;
         }
         Dc_Trapeze_pos(DC_PORT, position_depart_trapeze,3.1415*2+(COMPTEUR_TOUR*2*3.1415), 3.1415*11*2/(temps_boucle));

     }
     //Fin de la boucle
     else
     {
         Dc(0,DC_PORT);
         printf("position %lf \n", (rc_get_encoder_pos(DC_PORT)-DC_POSITION_DEPART_BRAS));
         antiWhile = 0;
         return 1;
     }
     return 0;
}
```

```c
//COMMMANDE TRAPEZE sans ASSERVISSEMENT EN POSITION
int Dc_Trapeze_pos(int DC_PORT, double DC_POSITION_DEPART, double DC_POSITION_DESIRE, double DC_VITESSE_PLATEAU)
{
   double  vitesseTrapeze;

  //ACCELERTION
  if ((rc_get_encoder_pos(DC_PORT)-DC_POSITION_DEPART)*2*3.1415/1500 <= DC_POSITION_DESIRE*0.33 )//[s]
  {
    vitesseTrapeze = DC_VITESSE_PLATEAU*3*(rc_get_encoder_pos(DC_PORT)-DC_POSITION_DEPART)*2*3.1415/1500/DC_POSITION_DESIRE;
  }

  //PLATEAU
  else if ((rc_get_encoder_pos(DC_PORT)-DC_POSITION_DEPART)*2*3.1415/1500 <= DC_POSITION_DESIRE*0.66 )
  {
    vitesseTrapeze = DC_VITESSE_PLATEAU;
  }

  //DESCENTE
  else if ((rc_get_encoder_pos(DC_PORT)-DC_POSITION_DEPART)*2*3.1415/1500 <= DC_POSITION_DESIRE)//[ms]
  {
    vitesseTrapeze = -3*(DC_VITESSE_PLATEAU*((rc_get_encoder_pos(DC_PORT)-DC_POSITION_DEPART)*2*3.1415/1500)/DC_POSITION_DESIRE-DC_VITESSE_PLATEAU);
  }

  //VERIFICATION SATURATION
  if (vitesseTrapeze < 1.5 && ((rc_get_encoder_pos(DC_PORT)-DC_POSITION_DEPART)*2*3.1415/1500 < DC_POSITION_DESIRE))
  {
    vitesseTrapeze = 1.5;
  }

  Dc(vitesseTrapeze,DC_PORT);

  return 1;

}
```

```
//GESTION JAMBE
int Servo_Jambe(int SERVO_PORT_JAMBE, int DC_EPAULE_PORT, int DC_POSITION_DEPART_EPAULE)
{

    //GESTION DE LA JAMBE
    if((rc_get_encoder_pos(DC_EPAULE_PORT) - DC_POSITION_DEPART_EPAULE)< tour_bras*0.25*rapport_engrenage)
    {
        Servo(3.1415,50,PORT_SERVO_JAMBE_G,2);
    }
    else if((rc_get_encoder_pos(DC_EPAULE_PORT)/rapport_engrenage -DC_POSITION_DEPART_EPAULE)>= tour_bras*0.25
         && (rc_get_encoder_pos(DC_EPAULE_PORT)/rapport_engrenage -DC_POSITION_DEPART_EPAULE)<  tour_bras*0.50)
    {
        Servo(0,50,PORT_SERVO_JAMBE_G,2);
    }
    else if((rc_get_encoder_pos(DC_EPAULE_PORT)/rapport_engrenage -DC_POSITION_DEPART_EPAULE)>= tour_bras*0.50
         && (rc_get_encoder_pos(DC_EPAULE_PORT)/rapport_engrenage -DC_POSITION_DEPART_EPAULE)<  tour_bras*0.75)
    {
        Servo(3.1415,50,PORT_SERVO_JAMBE_G,2);
    }
    else if ((rc_get_encoder_pos(DC_EPAULE_PORT) - DC_POSITION_DEPART_EPAULE)< tour_bras*rapport_engrenage)
    {
        Servo(0,50,PORT_SERVO_JAMBE_G,2);
    }

    return 0;
}
```

6.3 Implémentation

Etude nageur de crawl :
https://gregoirerichard99.wixsite.com/performancesnatation/blank-kr699

Coupleur d'axes :
https://www.gotronic.fr/art-coupleur-d-axes-8-8-mm-cl1m88-20808.htm

Joint spi :
https://www.123roulement.com/joints-OAS-8X14X5-NBR