Rapport de stage assistant ingénieur

Katell Lagattu

Robot nageur

Du 07/06/2021 au 31/08/2021

Lien Github du rapport technique

Voici le lien où sont rassemblés tous les codes, documents et fichiers CAO utilisés pendant le stage. Des fichiers README expliquent chaque partie du projet en détail.

`https://github.com/Katell-Lag/robot_nageur`

Remerciements

En premier lieu, je tiens à remercier mon maître de stage M. René ZAPATA, enseignant-chercheur au LIRMM. Je le remercie pour sa confiance et pour m'avoir proposé ce sujet de stage passionnant. Il m'a permis de découvrir le domaine de la recherche en robotique.

J'aimerais également remercier les enseignants de Polytech Montpellier M. Lionel LAPIERRE et M. Éric DUBREUIL pour avoir pris le temps de nous aider, ainsi que M. Olivier MOISE, technicien du bâtiment 14 –lieu de stage-, pour son soutien et son aide tout au long du stage.

Je remercie également M. Fabrice Le Bars, professeur à l'ENSTA Bretagne, qui s'est plongé dans le projet malgré la distance et qui nous a apporté son aide sur des difficultés techniques.

Je suis très reconnaissante d'avoir travaillé avec mes collègues de stage Mme Anna Agobian, M. Enzo RAFFINESQUE ainsi que M. Medhi TOUNDI. Merci pour le travail réalisé ensemble et la bonne entente dans notre équipe.

Table des matières

1 Introduction

La deuxième année d'école d'ingénieurs requiert un stage assistant ingénieur d'une durée de douze semaines. Étant une élève de l'ENSTA Bretagne en spécialité Robotique Autonome, et souhaitant découvrir le monde de la recherche, j'ai souhaité effectuer ce stage dans un laboratoire. J'ai donc postulé auprès de plusieurs laboratoires de robotique en France, et en particulier au LIRMM (Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier) dont les domaines de recherche m'intéressaient particulièrement. M. Zapata, Professeur des Universités au LIRMM, a accepté de m'intégrer à son équipe autour d'un projet : construire un robot nageur.

Mon stage a débuté le 7 juin 2021, et s'est terminé le 31 août. Au cours de ce stage, j'ai pu aborder différents aspects de la robotique : simulations d'un robot, modélisation 3D, assemblages mécaniques, électronique, informatique, commande et autonomie du robot. De plus, plusieurs compétences ont été mobilisées pour ce stage, comme la communication, la créativité et le travail en équipe. Après une brève présentation du LIRMM et une mise en contexte du projet, ce rapport détaillera ces différents aspects du stage.

2 Présentation du LIRMM et mise en contexte du projet

2.1 Présentation et contexte

Le Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier est un centre de recherche dépendant à la fois de l'Université de Montpellier et du Centre National de la Recherche Scientifique et est situé sur le Campus Saint-Priest. Les domaines de recherches du LIRMM concernent en particulier les systèmes embarqués, les études en algorithmique, la bio-informatique, les interactions homme-machine et la robotique. Le laboratoire est divisé en trois départements scientifiques de recherche : Informatique, Microélectronique et Robotique.

Mon tuteur de stage M. Zapata fait partie du département de Robotique et travaille au sein de l'équipe EXPLORE. Cette équipe s'intéresse à la robotique mobile pour les milieux terrestres, marins et aériens.

Le projet que j'ai intégré consiste à concevoir un robot humanoïde nageant le crawl. Pour des raisons de disponibilité du matériel, le stage s'est effectué dans les locaux de Polytech Montpellier. Le robot nageur avait déjà fait l'objet du projet semestriel de quatre étudiants de Polytech : Anna Agobian, Tatiana Dewildeman, Théo Lemaire et Paul Masson. Ils ont travaillé sur la simulation du mouvement de bras sous Matlab, la conception et la fabrication du corps du robot et la commande en position d'un bras articulé simplifié [[1]].

La nouvelle équipe réunie autour du robot nageur que j'ai intégrée était composée d'Anna Agobian, étudiante en stage de deuxième année à Polytech Montpellier (MEA4) qui avait déjà travaillé sur le projet, et d'Enzo Rafinesque et Mehdi Toundi, tous deux en stage de première année et également étudiants à Polytech Montpellier (MEA3). Le stage a été co-encadré par M. Zapata et M. Lapierre, qui est Maître de Conférences et fait également partie de l'équipe EXPLORE.

2.2 Cahier des charges du robot nageur

L'objectif final est d'obtenir un robot humanoïde autonome nageant le crawl dans la mer et effectuant un circuit autour de trois bouées.

Le robot nageur est composé d'un buste, de deux bras et deux jambes ainsi que d'une tête. La tête du robot nageur contient notamment une caméra et un procédé de traitement d'images qui lui permet de reconnaître les bouées et donc de se localiser. Cette partie du projet a déjà été effectuée.

Chaque bras du robot nageur contient 3 moteurs, qui modélisent chacun une articulation. Le bras humain possède plusieurs degrés de liberté (voir figure 1).

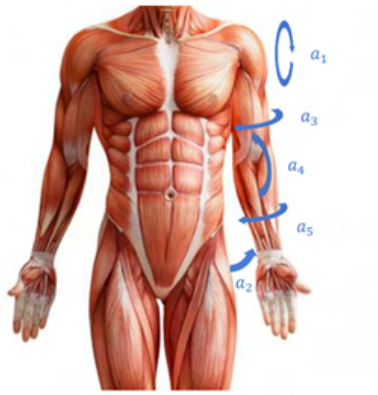


FIGURE 1 – Degrés de liberté du corps humain

Le moteur « épaule » correspond au degré de liberté a1, le moteur « bras » au degré de liberté a3 et le moteur « avant-bras » au degré de liberté a4.

Le buste du robot est composé de 3 tubes étanches. Le tube central (voir figure 2) contient la partie électronique du robot, et les deux autres tubes forment les épaules du robot. Ils contiennent chacun un moteur (qui simule le degré de liberté de l'épaule) et un encodeur associé (voir figure 3).

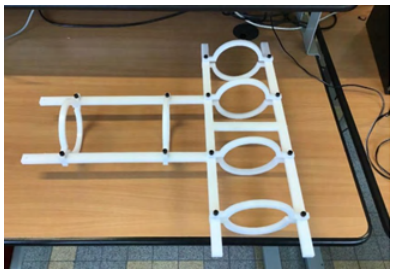


FIGURE 2 – Structure du corps

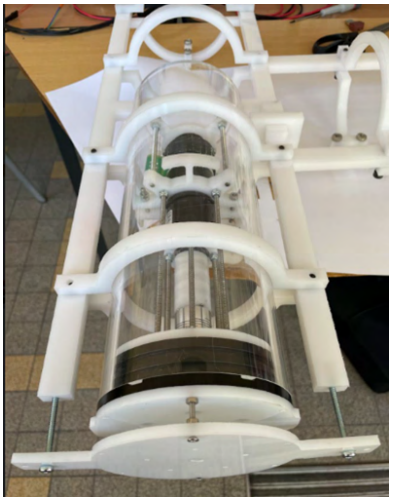


FIGURE 3 – Architecture de l'épaule obtenue à la fin du stage

Plusieurs solutions ont été envisagées pour les bras du robot : regrouper les deux moteurs au même endroit, ou les mettre l'un à la suite de l'autre. Nous verrons par la suite quelle solution est à privilégier. Au-delà de l'aspect humanoïde recherché, une rotation de 360° de l'articulation a3 est nécessaire pour dérouler les fils électriques reliant les moteurs et les cartes électroniques présentes dans le buste. En ce qui concerne les jambes, un système simple de battements de jambes a été imaginé.

Les problématiques sont notamment la flottabilité, la stabilité et la commande du robot. En effet, la houle peut déstabiliser le robot et les mouvements des bras peuvent rapidement se désynchroniser. La commande envisagée est une commande hybride position-couple [2].

Un autre point important est la mise en place des capteurs dans des tubes étanches parfois très petits.

Les enjeux du projet se trouvent donc dans la réalisation d'une architecture stable et dans la commande des bras à l'aide de capteurs.

2.3 Point de départ et objectifs

Le projet se découpe en plusieurs étapes qui sont décrites dans la figure 4.

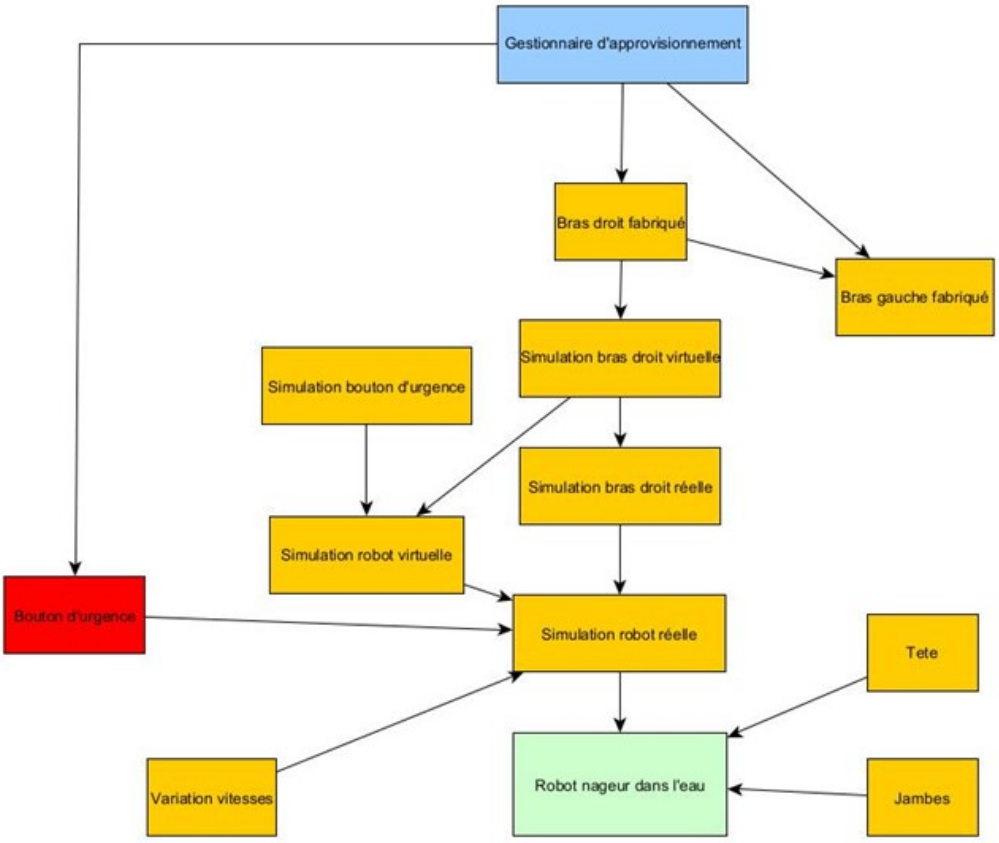


FIGURE 4 – Description des étapes du projet

Le premier groupe de stagiaires qui a travaillé sur ce sujet a pu définir une bonne trajectoire des bras du robot grâce à une simulation Matlab. Ils ont également réalisé la structure du buste et réalisé une première maquette d'un bras, qu'ils ont commandée en boucle ouverte (voir vidéo : `https://youtu.be/kqU7eqa2u2k`).

Enzo Rafinesque et Mehdi Toundi ont été chargés de la mise en place des capteurs (encodeurs et IMU). Avec Anna Agobian, nous avons eu pour mission de réaliser le montage des bras du robot et de commander les moteurs en boucle fermée.

Au cours du stage, nous avons simulé le robot, réalisé un bras ainsi que le bouton d'urgence, et implémenté la commande de deux manières différentes : avec une BeagleBone Blue (équivalent d'une RaspberryPi) et avec une carte Trinamic capable de jouer le rôle d'un ESC (Electronic Speed Control) et de contrôler le moteur.

3 Simulation du robot avec V-REP

3.1 Simulation de l'architecture

L'architecture du robot créé sous V-REP est celle du vrai robot : un buste, deux bras, deux jambes. Les articulations sont modélisées par des joints cylindriques et les parties mécaniques par des cylindres. Le buste est fixe et modélisé par un pavé droit. C'est la première pièce dans l'arbre des dépendances (voir figure 5).

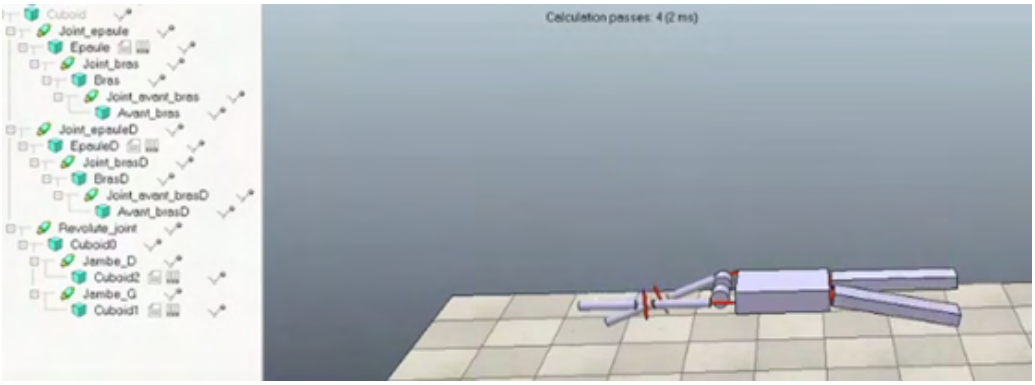


FIGURE 5 – Modèle du robot sur V-REP

3.2 Commande en position

Les 8 moteurs (3 par bras et 2 pour les jambes) doivent être commandés en position. Avec Anna, nous avons récupéré le tableau de valeurs de l'ancien groupe, qui correspond aux angles que doivent atteindre chaque pivot à chaque instant. Le programme a été fait en Lua. En adaptant les unités et les repères, les 6 moteurs ont pu être commandés en position grâce au tableau de valeurs (voir figure 6) :

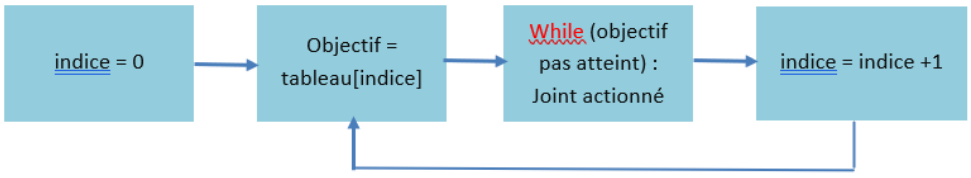


FIGURE 6 – Schéma de la commande du robot sur V-REP

Lien vidéo vers une première simulation : `https://youtu.be/ZyJ85cyOGq4`

Nous avons également voulu simuler le milieu aqueux sous V-REP en ajoutant des frottements lorsque le bras était censé être dans l'eau [3]. Nous avons également testé différents paramètres comme la densité des pièces. La réaction du bras sous V-REP était alors imprévisible et peu réaliste, et nous avons laissé cette piste de côté.

3.3 Autre méthode utilisée pour obtenir un tableau de valeurs d'angles

La commande en position sous V-REP est basée sur le tableau de valeurs contenant les angles que les pivots doivent atteindre au cours du temps. Pendant les trois semaines en distanciel du mois d'août lorsque le laboratoire était fermé, j'ai travaillé sur une librairie Python spécialisée dans la reconnaissance de mouvements, (voir figure 7). Cette librairie utilise OpenCV et permet en particulier de donner les positions des différentes articulations de la personne en face de la webcam.

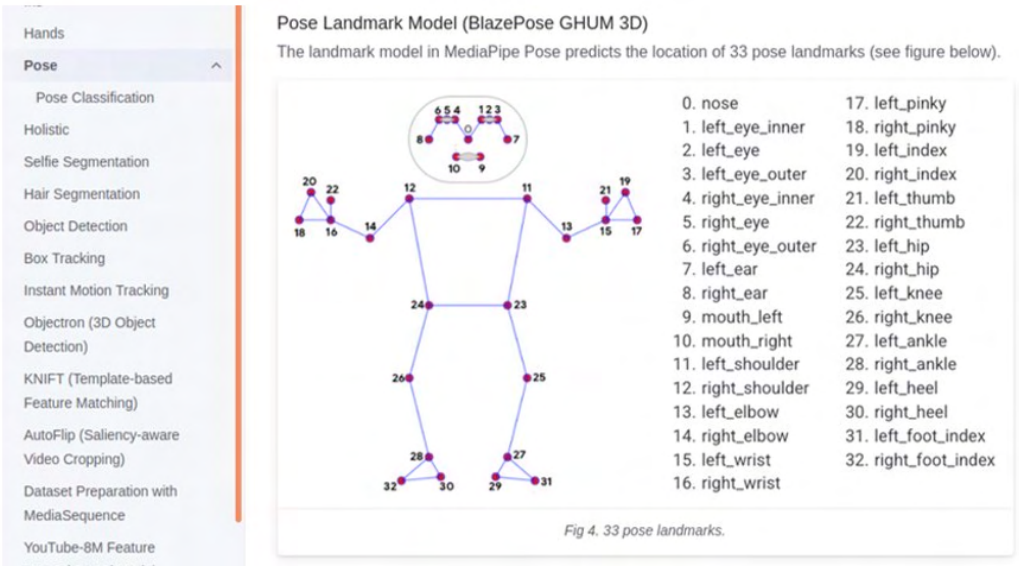


FIGURE 7 – Points de l'espace repérés par la librairie MediaPipe

J'ai donc utilisé ce module pour récupérer les positions des articulations 11, 12, 13, 14, 15 et 16 lorsque je mimais le mouvement de crawl devant la caméra de mon PC (voir figure 8).

FIGURE 8 – Capture d'écran d'un test de MediaPipe

Une fois ces positions enregistrées, j'ai calculé les angles qui m'intéressaient avec la formule d'Al-Kashi, ou théorème de Pythagore généralisé (voir équation (1)). Un extrait du code est détaillé figure 9.

Le théorème d'al-Kashi s'énonce de la façon suivante : Soit un triangle ABC, dans lequel on utilise ces notations : d'une part α, β et γ pour les angles et, d'autre part, a, b et c pour les longueurs des côtés respectivement opposés à ces angles.

$$c^2 = a^2 + b^2 - 2abcos(\gamma) \tag{1}$$

```
if results.pose_landmarks:
    #mpDraw.draw_landmarks(img, results.pose_landmarks)
    mpDraw.draw_landmarks(img, results.pose_landmarks,mpPose.POSE_CONNECTIONS)
    #extraction de chaque marques
    for id, lm in enumerate(results.pose_landmarks.landmark):
        h, w, c = img.shape
        cx, cy = int(lm.x * w), int(lm.y * h)
        #cv2.circle(img,(cx,cy),5, (0,255,255),cv2.FILLED)

cTime = time.time()
fps = 1/(cTime-pTime)
pTime = cTime
cv2.putText(img, str(int(fps)), (70, 50), cv2.FONT_HERSHEY_PLAIN, 3, (255, 255, 0), 3)
cv2.imshow("Image", img)
cv2.waitKey(1)
k +=1
P1x = results.pose_landmarks.landmark[13].x
P1y = results.pose_landmarks.landmark[13].y
P2x = results.pose_landmarks.landmark[11].x
P2y = results.pose_landmarks.landmark[11].y
P3x = results.pose_landmarks.landmark[15].x
P3y = results.pose_landmarks.landmark[15].y

P12 = np.sqrt((P1x-P2x)**2 + (P1y-P2y)**2)
P13 = np.sqrt((P1x-P3x)**2 + (P1y-P3y)**2)
P23 = np.sqrt((P3x-P2x)**2 + (P3y-P2y)**2)

angle = np.arccos( (P12**2 + P13**2 - P23**2) / (2*P12*P13) )
angledegre = angle*180/np.pi
if i%3 ==0:
    fichier.write(str(angle))
    fichier.write("\n")
print(angledegre)
i = (i+1)%3
```

FIGURE 9 – Partie du code python réalisé : on voit ici l'utilisation du théorème d'Al-Kashi

J'ai également travaillé sur la synchronisation des bras, et modélisé les mains du robot, qui sont un élément essentiel de la nage en crawl.

Lien vers la simulation V-REP obtenue, plus réaliste : `https://youtu.be/ThywfFByKGQ`

On voit sur la vidéo que l'entrée du bras dans l'eau est plus fluide et plus humanoïde. Le coude se plie à la remontée du bras dans un soucie de réalisme. La main du robot est également commandée à l'aide d'un tableau de valeurs d'angles.

4 Partie Hardware

4.1 Fabrication du robot

Le premier groupe qui a travaillé sur ce projet avait fixé le moteur de l'épaule à un support et avait mis en place l'encodeur (voir figure 10). Avec Anna nous avons donc eu pour mission de réaliser le montage final.

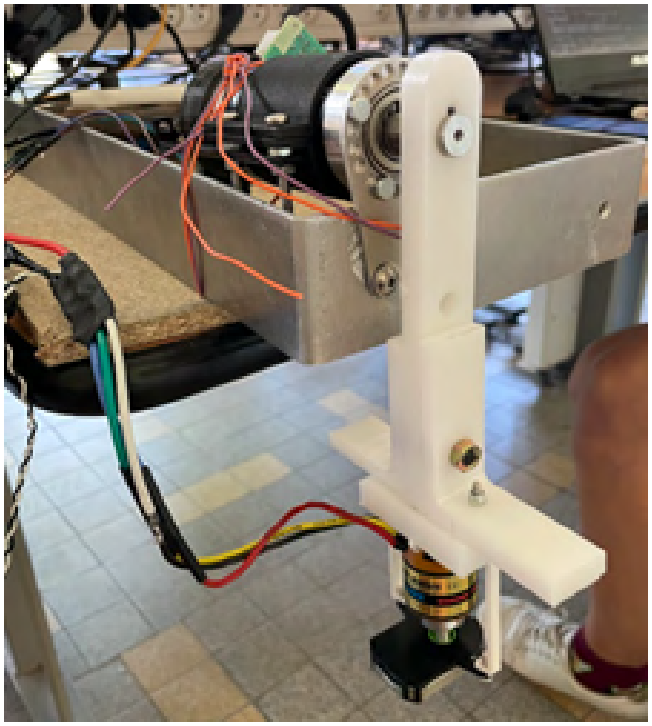


FIGURE 10 – Montage du groupe précédent : il s'agit du moteur épaule

L'épaule du robot finale est constituée d'un tube étanche contenant le moteur, l'encodeur et leurs supports. L'axe du moteur passe par une pièce transitoire et un coupleur d'axe, et ressort du tube étanche en passant par une bride. L'étanchéité sera par la suite assurée par une mise sous vide et des câbles étanches entre les tubes, passant par des pénétrateurs. Le tube de l'épaule doit également être solidement fixée à la structure du buste du robot, et l'on doit à tout prix éviter le jeu à l'intérieur du tube pour que la rotation du moteur soit homogène. Enfin, les cartes électroniques doivent être solidement fixées à l'intérieur du tube principal (voir figure 11).

Nous avons passé trois semaines à réaliser un montage stable et fonctionnel d'un premier bras. La principale difficulté était la fixation du moteur dans le tube. Il a fallu imaginer des pièces originales permettant de stabiliser le moteur et l'encodeur, par exemple la pièce à oreillettes (voir figure 12). Nous faisions nos tests avec un poids accroché à l'axe afin de se rendre compte de la torsion de la structure dans le tube.

Par exemple, une première idée était de relier avec des même tiges filetées le support de l'encodeur, la pièce à oreillette qui fait office de support de moteur et la pièce proche de la sortie du tube. Cette configuration n'était pas bonne car un couple trop grand était exercé sur les tiges filetées, ce qui les vrillait et engendrait des vibrations. Une solution était de séparer les tiges filetées en deux : certaines ont servi à fixer le support de l'encodeur sur le support du moteur, et d'autres à fixer le support du moteur à la pièce de la sortie. Cette structure est beaucoup plus stable.

FIGURE 11 – Support de la BeagleBone Blue, fait à l'imprimante 3D

Une autre difficulté était l'alignement des tiges filetées qui supportent la structure, et l'encastrement des différentes pièces à l'aide d'écrous-stop. Il fallait en effet que les tiges filetées soient parfaitement alignées pour éviter les torsions. Les écrous devaient aussi être placés de manière très précise pour assurer une certaine distance entre chaque pièce.

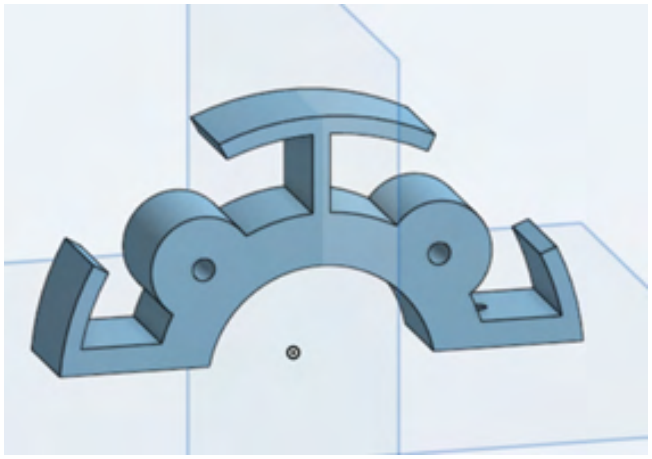


FIGURE 12 – "Pièce à oreillettes" dans le logiciel de CAO OnShape

Nous avons également eu à réfléchir aux dimensionnements des pièces et à la taille des différents trous. Nous avons souvent dû refaire plusieurs fois nos pièces en modifiant le détail d'un trou ou d'une épaisseur.

Nous avons principalement utilisé une fraiseuse CNC et une imprimante 3D pour réaliser nos pièces. La modélisation 3D a été réalisée avec le logiciel OnShape. Nous avons également usiné les pièces avec des perceuses et des outils à tarauder et à poncer. Il a aussi fallu souder de nombreux fils et composants.

Finalement, nous avons réussi à réaliser une structure stable et performante (voir figure 13). L'architecture de l'épaule gauche est validée, et nous avons sauvegardé tous les fichiers CAO en vue de la réalisation de l'épaule droite.

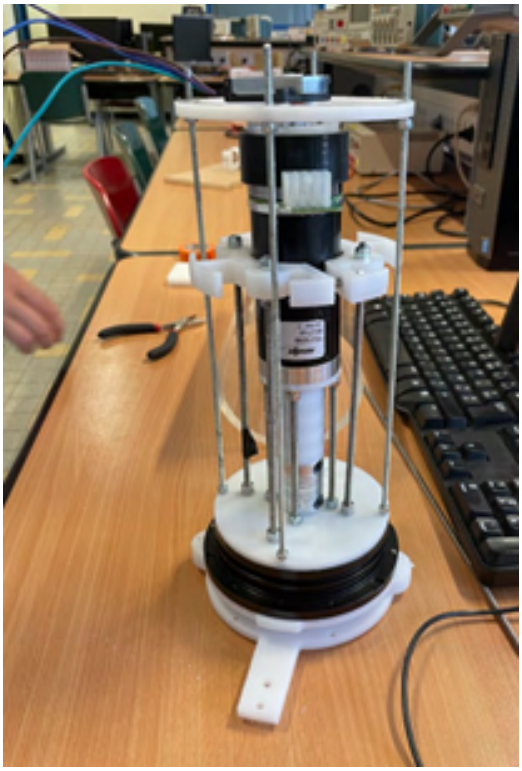


FIGURE 13 – Structure complète de l'épaule du robot

Concernant le bras et l'avant-bras du robot, M. Zapata avait d'abord réalisé un prototype composé d'un axe couplé à l'axe du moteur, d'un tube et de deux moteurs côte-à-côte assurant les degrés de libertés du bras et de l'avant-bras. Après plusieurs tests qui seront détaillés par la suite, nous avons conclu que cette structure n'était pas adaptée : les deux moteurs étaient trop éloignés de l'axe du moteur, ce qui engendrait un bras de levier important. Cela avait pour conséquence de décaler les engrenages de l'avant-bras, ce qui l'empêchait de tourner correctement. De plus, le moteur de l'épaule rencontrait des difficultés à rester fixe dans son tube lors de la descente du bras.

Une piste d'amélioration serait de réaliser un second prototype avec les deux moteurs l'un à la suite de l'autre. Cependant, dans le cadre de ce stage, tous nos tests ont été réalisés sur la première version du bras avec les deux moteurs côte-à-côte.

Toutes les mises en plan des pièces réalisées sur le logiciel OnShape et ensuite imprimées ou découpées à la CNC sont en annexe 6.

4.2 Le bras du robot

Le bras et l'avant-bras du robot nageur sont constitués de trois axes et trois rotations. Son schéma est présenté figure 14 (les longueurs L1, L2 et L3 sont fixées à 17cm, 27cm et 19 cm).

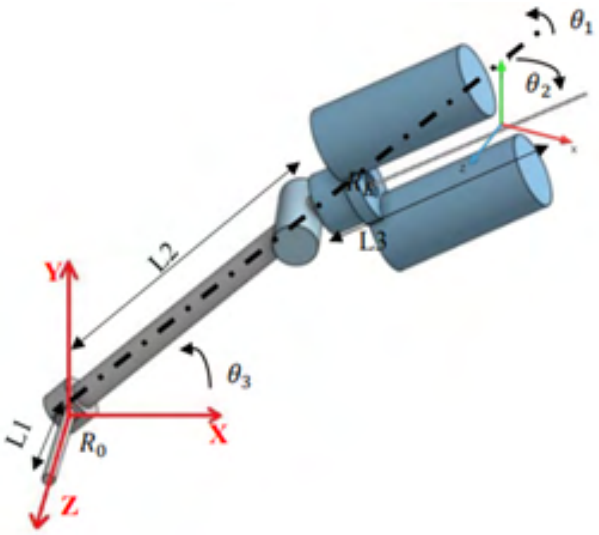


FIGURE 14 – Schéma du bras du robot

Les trois degrés de liberté sont assurés par trois moteurs. Le moteur 1 assure la rotation de 360° de l'épaule (θ_3) Les moteurs 2 et 3 assurent la rotation du bras (θ_1) et le pliage du coude (θ_2) Pour obtenir une vision plus concrète de ce mécanisme, le modèle d'actionnement et le modèle géométrique directe de notre robot sont décrits par les équations (2) à (5) [4].

- Modèle d'actionnement :

$$\begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix} = \begin{pmatrix} 1/60 & 0 & 0 \\ 1/60 & -1/60 & 0 \\ 0 & 0 & 1/756 \end{pmatrix} * \begin{pmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \end{pmatrix} \tag{2}$$

δ_1, δ_2 *et* δ_3 correspondent aux positions angulaires des trois moteurs. Les coefficients de la matrice 3x3 correspondent aux rapports de réduction.

- Modèle de géométrie directe :

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} L_2 cos(\theta_3) + L_3 cos(\theta_2 + \theta_3) \\ L_2 sin(\theta_3) + L_3 sin(\theta_2 + \theta_3) \\ L_1 \end{pmatrix} \tag{3}$$

Ce système nous permet donc de connaitre la position en x, y, z du bout du bras en fonction de la position angulaire des moteurs. Afin d'obtenir ces positions angulaires en fonction des coordonnées il faut inverser ce système :

$$\theta_2 = \pm arccos(\frac{x^2 + y^2 - L_2^2 - L_3^2}{2L_2L_3}) \tag{4}$$

$$\theta_3 = arccos(\frac{x(L_2 + L_3 cos(\theta_2)) + yL_3 sin(\theta_2)}{x^2 + y^2}) \tag{5}$$

4.3 Capteurs et bouton d'arrêt

Les capteurs qui seront utilisés pour pouvoir réaliser une commande hybride position-couple sont des encodeurs et des centrales inertielles (IMU). Il y aura 6 encodeurs et 4 IMU pour contrôler les bras du robot : un encodeur pour chaque moteur, et 2 IMU par bras (une sur le bras et l'autre sur l'avant-bras). Le repère de référence du robot sera donné soit par une

cinquième IMU, soit par l'IMU intégrée d'une BeagleBone Blue si cette carte est utilisée. Le groupe d'Enzo Rafinesque et Mehdi Toundi a travaillé sur ces capteurs. La principale difficulté était d'intégrer les encodeurs à la structure en respectant des critères de taille. Ils ont également géré les données des IMU avec une carte STM32.

Par souci de sécurité, M. Zapata nous a demandé de mettre en place un bouton d'urgence qui coupe l'alimentation des moteurs (voir figure 15).

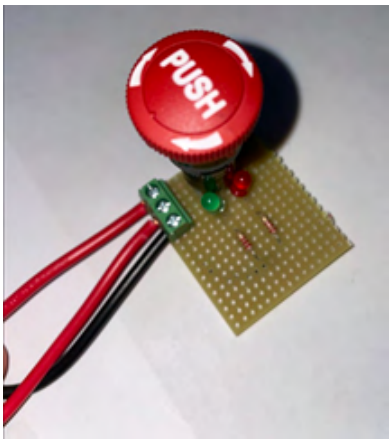


FIGURE 15 – Bouton d'arrêt d'urgence soudé sur plaque (led verte allumée quand le moteur tourne, led rouge allumée quand le bouton est activé

4.4 Architecture électronique

Le choix de l'électronique de ce robot n'est pas encore définitif. En effet, nous sommes actuellement capables de réaliser le mouvement et de commander le robot via la BeagleBone Blue, mais un des objectifs du projet est d'utiliser une carte électronique spécialement conçue pour la commande hybride : une Trinamic.

En effet, la difficulté de la commande est que le bras évolue à la fois dans l'air et dans l'eau. Ainsi, réaliser une nage crawlée nécessite une adaptation du couple appliqué suivant la position du bras dans son environnement.

Le couple appliqué sous l'eau doit être plus important qu'en dehors si l'on souhaite obtenir une bonne trajectoire et un mouvement constant. Pour cela il faut utiliser un système de commande en couple et en position (hybride). C'est ce que devrait permettre de faire la carte Trinamic que nous avons étudiée.

L'annexe 1 montre un exemple d'architecture électronique si l'on utilisait la BeagleBone Blue. Réaliser un schéma d'architecture électronique est important, notamment pour avoir une idée du nombre de câbles et donc de pénétrateurs dans les tubes à prévoir.

5 Partie Software

5.1 Avec une BeagleBone Blue

5.1.1 Connectique

La carte BeagleBone Blue (BBBlue) est une carte de développement open source fonctionnant sous Debian. Elle dispose de divers modules et interfaces (UART, SPI, IMU, Bluetooth) ainsi que de connecteurs moteurs et encodeurs (voir figures 16 et 17).

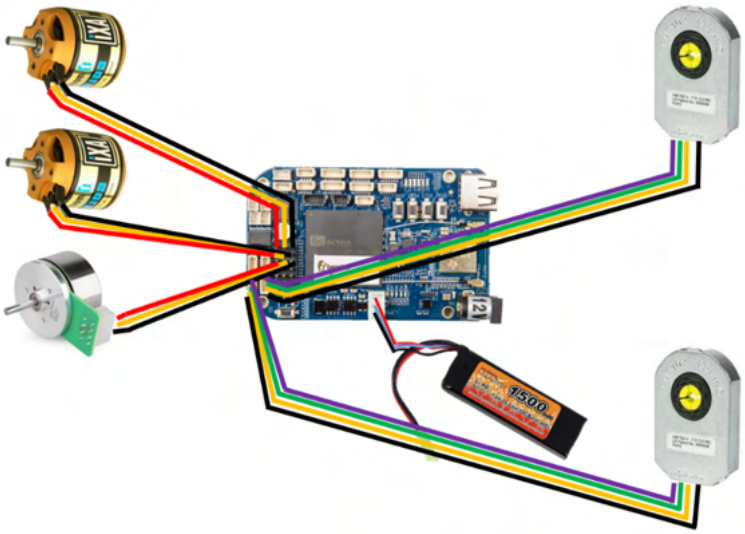


FIGURE 16 – Connectique entre la BBlue, les trois moteurs, les encodeurs et l'alimentation

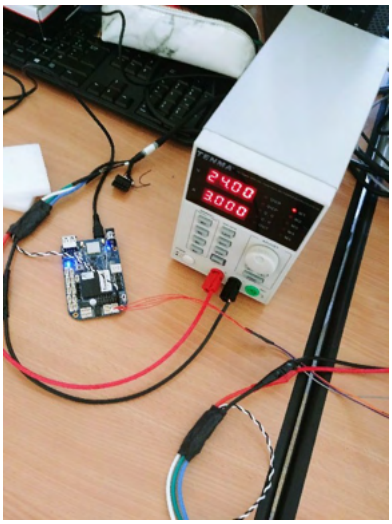


FIGURE 17 – Banc de test de la BBlue

Puisque la carte possède 8 connecteurs moteurs et 4 connecteurs encodeurs, nous avons choisi de commander 3 moteurs BLDC (brushless à courant continu) avec une unique BBBlue. Il sera donc nécessaire de disposer de 2 BBBlues pour commander l'ensemble des moteurs.

5.1.2 Commande

Concernant la commande en position, il s'agit d'une commande en boucle ouverte codée en langage C. Elle repose sur les valeurs des encodeurs. Le bras que nous avons étudié était composé d'un moteur au niveau de l'épaule, et de deux autres moteurs (pour les degrés de liberté du bras et de l'avant-bras) situés côte-à-côte au niveau du coude du robot.

Le degré de liberté du bras était obtenu en faisant tourner les deux moteurs en même temps dans des sens opposés, et le degré de liberté de l'avant-bras était obtenu en faisant tourner seulement un des moteurs (voir équation (1)).

Nous avons d'abord exécuté le code avec le bras et l'avant-bras du robot séparés du corps. Le mouvement était correct. Voici un lien vidéo montrant le mouvement de crawl des éléments du bras non assemblé : `https://youtu.be/XoL0Z7ITHHk`

Nous avons ensuite monté le bras sur le corps du robot, et relancé le code (voir figure 18). Le résultat n'était pas bon : le poids des moteurs était trop important et empêchait le bon fonctionnement du système. Les dents des engrenages n'étaient pas assez alignées et le corps du robot subissait une grosse flexion. Voici le lien vidéo de notre essai : `https://youtu.be/9AdFpL0x2fE`

On voit dans la vidéo que les moteurs sont trop lourds, empêchant de réaliser le mouvement correctement.

FIGURE 18 – Assemblage du bras et de l'épaule du robot nageur

Nous avons alors renforcé la structure en ajoutant des pièces de soutien au moteur, mais ce n'était pas suffisant. Avec notre tuteur, nous en avons conclu que toute l'architecture du bras était à revoir. Le bras devrait plutôt être composé de deux moteurs mis l'un à la suite de l'autre.

Nous avons également codé une commande en vitesse en boucle fermée. Pour cela nous avons implémenté un contrôleur proportionnel qui régule l'intervalle PWM de façon à avoir des différences de valeurs d'encodeur constantes. En effet, l'erreur correspond à :

((valeur_actuelle_encodeur - valeur_ancienne_encodeur) - (valeur_de_reference)).

Le résultat obtenu était partiellement satisfaisant : le bras avait une vitesse plus homogène (notamment lors du passage entre la montée et la descente du bras) mais il tournait par petits à-coups.

5.1.3 Parallèle avec la simulation Matlab et V-REP

Pour commander le robot, nous nous sommes basées sur le même tableau de valeurs que nous avons utilisé dans V-REP. Ce tableau de valeurs avait été obtenu par l'ancien groupe grâce à une simulation Matlab, dont le principe est décrit figure 19 :

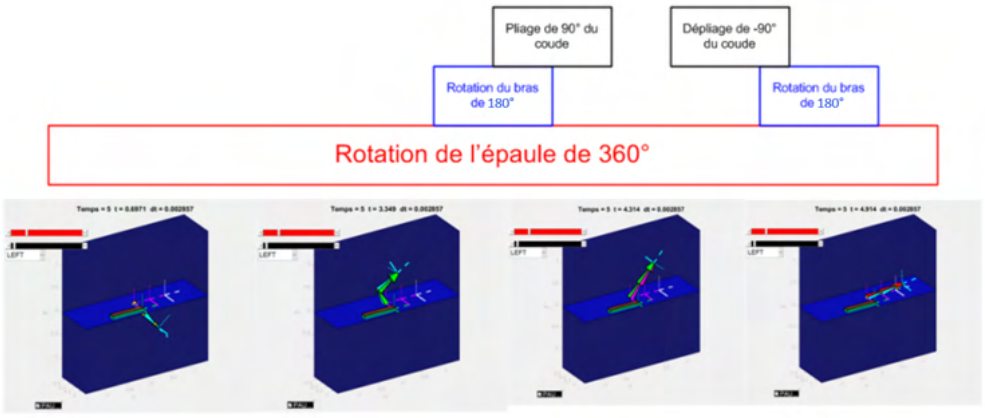


FIGURE 19 – Etapes du mouvement du crawl

La différence avec V-REP est que l'on ne donne pas pour consigne les valeurs du tableau les unes à la suite des autres. On actionne les moteurs selon les valeurs des encodeurs. Nous avons enregistré ces valeurs pour comparer avec le schéma de Matlab, et pour vérifier que le mouvement est bien le même (voir figure 20). Une partie du code C utilisé est donnée en annexe 2.

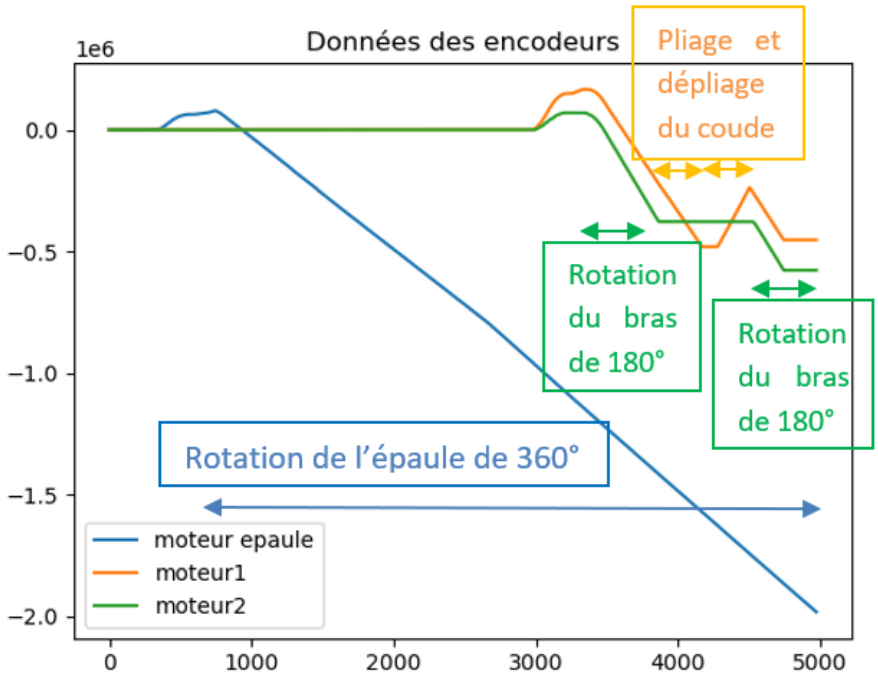


FIGURE 20 – Courbes des encodeurs

On voit sur ce graphique que les courbes des trois encodeurs correspondent bien au mouvement de crawl :

- Le moteur épaule tourne de 360°
- Les moteurs 1 et 2 s'actonnent en même temps pour la rotation du bras
- Le moteur 1 s'actionne seul pour le pliage et dépliage du coude

Nous avons donc obtenu un mouvement similaire à celui modélisé sur V-REP.

5.2 Avec une Trinamic

5.2.1 Rôle de la carte Trinamic

Au cours du stage, il nous a été demandé de travailler avec une carte électronique Trinamic TMC4671+TMC6100-BOB. C'était la première fois que le laboratoire commandait ce type de carte électronique et notre projet de robot nageur était l'occasion de la tester.

Les moteurs utilisés dans notre projet sont des moteurs brushless, difficilement commandables notamment à faible vitesse. Nous avons ainsi vite remarqué des défauts de la rotation lorsque nous commandions ces moteurs via la BeagleBone et un simple ESC. La carte Trinamic joue le rôle de l'ESC, mais contient également des algorithmes de contrôles : grâce à cette carte, le moteur est contrôlé via un FOC (Field Oriented Control). La Trinamic gère également le retour des encodeurs et peut ainsi commander le moteur en boucle fermée. Elle est donc capable de faire une commande en position et en couple.

L'architecture électronique imaginée en utilisant cette carte est visible en annexe 3. Il faudrait une carte Trinamic par moteur.

Afin d'étudier cette carte TMC4671+TMC6100-BOB, nous avions à notre disposition un kit de développement TMC4671+TMC6100-eval-kit. Ce kit permet de décomposer la carte pour mieux comprendre son fonctionnement. Pour la faire fonctionner, l'entreprise Trinamic a conçu sa propre interface.

5.2.2 La commande FOC

La commande FOC, appelée également "commande vectorielle", permet de garder une précision continue des commandes envoyées, notamment pour les moteurs BLDC. Pour se faire, c'est à partir de la consigne de vitesse du moteur que le flux et le couple nécessaires sont calculés et qu'ensuite les courants requis sont déduits. La commande FOC est composée de deux forces Q (quadrature) et D (direct) : l'une perpendiculaire et l'autre parallèle à l'axe des pôles du rotor. Le fonctionnement du FOC est détaillé sur la figure 21 :

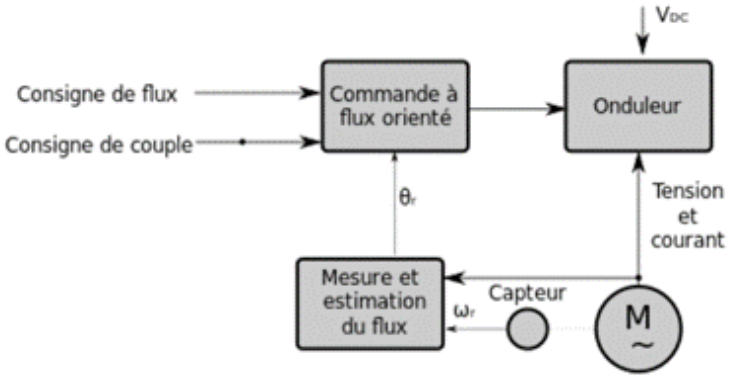


Figure 21 – Modèle de fonctionnement du FOC

5.2.3 Utilisation de la TMC4671+TMC6100-eval-kit

La TMC4671+TMC6100-eval-kit est composée d'un microcontrôleur Landungsbrücke, de deux cartes de liaisons Eselsbrücke, d'une carte "motion controller" TMC4671-EVAL et d'une carte driver TMC6100-EVAL (voir figure 22).

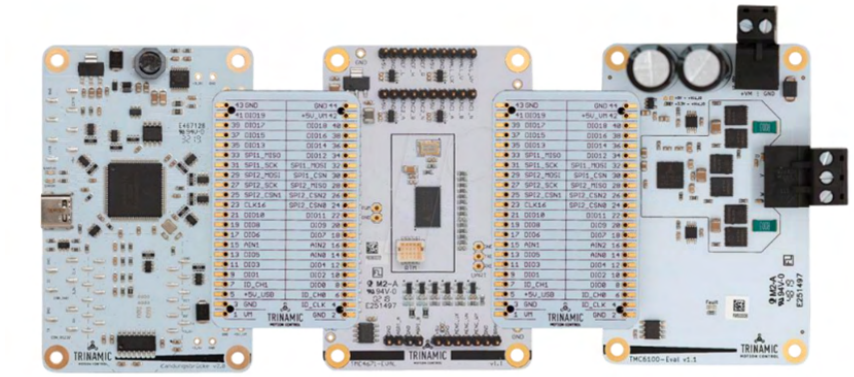


FIGURE 22 – Carte Trinamic TMC4671+TMC6100-eval-kit

Trinamic est une entreprise spécialisée dans la création de solutions de contrôle de mouvement. Elle conçoit des cartes électroniques qui optimisent le mouvement et le contrôle des moteurs. Les ingénieurs de Trinamic ont créé leur propre interface appelée « TMCL-IDE » -Trinamic Motion Control Language- Integrated Development Environment- afin de permettre de développer rapidement des applications utilisant leurs produits, et dans notre cas la TMC4671+TMC6100-eval-kit. En effet, il permet de visualiser en direct le comportement du moteur BLDC en fonction des encodeurs, de modifier les coefficients PID (Proportionnel, Intégral, Dérivé) du correcteur et tous les paramètres de la carte (voir annexe 4).

Il est également possible de changer le firmware de la carte mère Landungsbrücke afin d'y entrer par défaut les paramètres souhaités. Il s'agit en fait d'écrire des données dans les registres de la carte. Le TMCL-IDE permet de convertir les paramètres entrés dans l'interface en un code C contenant ces écritures de registres (voir annexe 5).

Le principal objectif de ce kit était de nous permettre de comprendre le fonctionnement de ces cartes, de visualiser la différence entre la commande en boucle ouverte/boucle fermée, et de s'initier à la commande des cartes Trinamic en vue d'utiliser la carte compacte TMC4671+TMC6100-Bob.

Voici les essais que nous avons réalisés et les résultats que nous avons obtenus :

• La commande directe en passant par le TMCL-IDE fonctionne. On peut voir le schéma du montage sur la figure 23. Nous avons pu commander notre moteur en couple et en boucle fermée. La commande FOC fonctionne également. À partir de ce résultat concluant, nous avons exporté les paramètres modifiés, afin de les insérer dans le code C du firmware. Nous avons essayé de l'importer dans la carte mais sans succès [[5]]. Nous ne pouvions plus communiquer avec notre kit et il nous a donc fallu réinitialiser la carte mère par court-circuit. Nous n'avons actuellement pas réussi à résoudre ce problème de changement de firmware, malgré notre prise de contact avec le service technique de Trinamic.

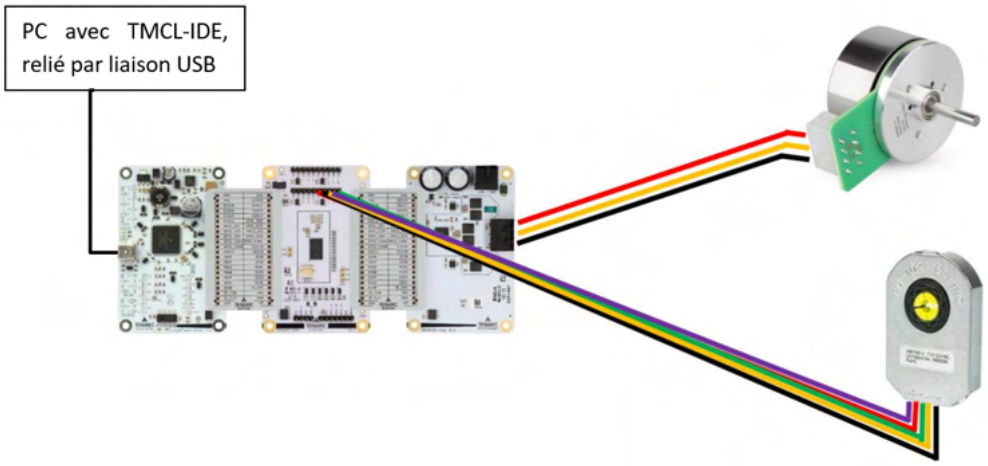


FIGURE 23 – Connectique du kit de développement

La vidéo de la commande du moteur par TMCL-IDE est disponible ici : `https://youtu.be/5De3pveBqN8`

• Sans passer par l'IDE mais en exécutant un code python que nous avons réalisé (figure 24) nous parvenons à commander le moteur en boucle ouverte. Pour faire ce code nous avons utilisé la bibliothèque PyTrinamic [[6]]. Ce code fonctionne avec une liaison USB, entre la carte Trinamic et un PC ou entre la carte Trinamic et une Raspberry Pi3.

```
# Init encoder (mode 0)
TMC4671.writeRegister( TMC4671.registers.MODE_RAMP_MODE_MOTION, 0x00000008)
TMC4671.writeRegister( TMC4671.registers.ABN_DECODER_PHI_E_PHI_M_OFFSET, 0x00000000)
TMC4671.writeRegister( TMC4671.registers.PHI_E_SELECTION, 0x00000001)
TMC4671.writeRegister( TMC4671.registers.PHI_E_EXT, 0x00000000)
TMC4671.writeRegister( TMC4671.registers.UQ_UD_EXT, 0x000007D0)
time.sleep(6)
TMC4671.writeRegister( TMC4671.registers.ABN_DECODER_COUNT, 0x00000000)

# Feedback selection
TMC4671.writeRegister( TMC4671.registers.PHI_E_SELECTION, 0x00000003)
TMC4671.writeRegister( TMC4671.registers.VELOCITY_SELECTION, 0x00000009)

# Switch to torque mode
print("switch to torque mode...")
TMC4671.writeRegister( TMC4671.registers.MODE_RAMP_MODE_MOTION, 0x00000001)

" show abn_decoder_count "
print(TMC4671.readRegister(TMC4671.registers.ABN_DECODER_COUNT))

# Rotate right
print("---------------------------\n rotate right... \n ---------------------------")
TMC4671.writeRegister( TMC4671.registers.PID_TORQUE_FLUX_TARGET, 0x03E80000)
time.sleep(6)
for i in range(10):
    " show abn_decoder_count "
    time.sleep(1)
    print(TMC4671.readRegister(TMC4671.registers.ABN_DECODER_COUNT))
time.sleep(1)
# Rotate left
print("---------------------------\n rotate left... \n ---------------------------")
TMC4671.writeRegister( TMC4671.registers.PID_TORQUE_FLUX_TARGET, 0xFC180000)
time.sleep(6)
for i in range(10):
    " show abn_decoder_count "
    time.sleep(1)
    print(TMC4671.readRegister(TMC4671.registers.ABN_DECODER_COUNT))
time.sleep(1)
```

FIGURE 24 – Extrait du code python fonctionnel

Pour réaliser ce code, nous nous sommes inspirés d'un code trouvé sur le github de Trinamic [[7]] et nous l'avons adapté à notre carte et à notre moteur. Il faut par exemple régler le bon nombre de pôles du moteur, la vitesse maximale etc.

Une autre partie du travail a été de configurer une Raspberry à partir d'une carte micro SD vierge et de mettre en place un réseau : dans notre cas, la Raspberry Pi3 créée son propre hotspot wifi grâce à l'outil RaspAp Webgui (voir figure 25).

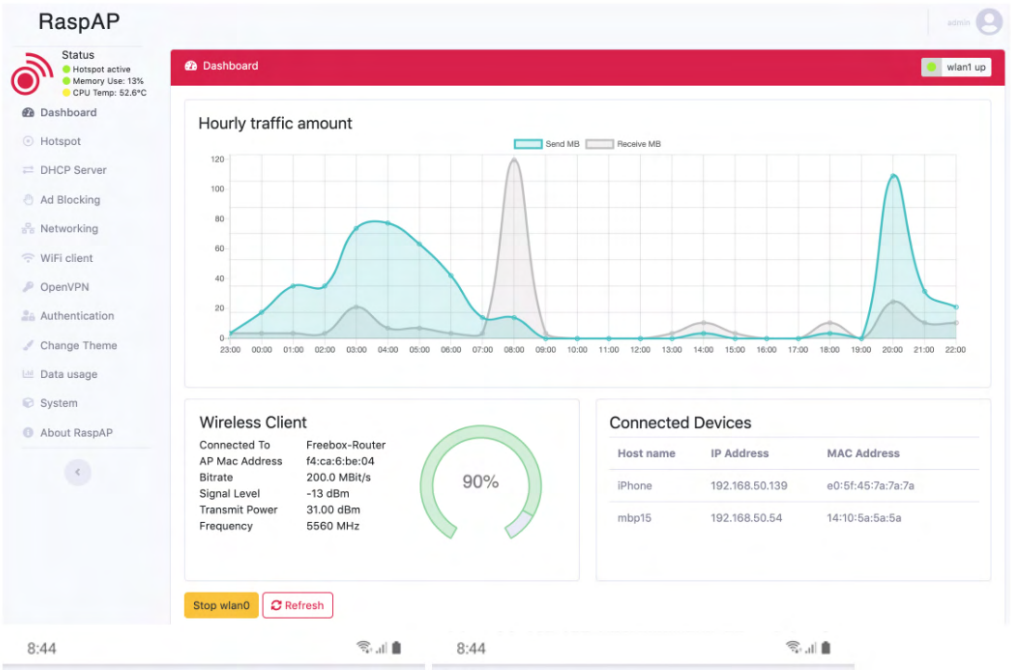


FIGURE 25 – Interface de RaspAp Webgui

Une fois que ces tests ont été fonctionnels, nous avons voulu passer à la version miniaturisée de la carte, et donc à une communication SPI (Serial Peripheral Interface).

5.2.4 Utilisation de la TMC4671+TMC6100-BOB

La carte Trinamic TMC4671+TMC6100-BOB est la version miniaturisée de la carte Trinamic TMC4671+TMC6100-eval-kit (voir figure 26). Cette carte permet de contrôler un moteur, de récupérer les données d'encodeurs et surtout de réaliser une boucle fermée. Le contrôle en boucle fermée de cette carte permet d'obtenir un système plus performant, stable et optimal car le retour des encodeurs permet d'asservir le système via un contrôleur PID. Elle assure au système une certaine stabilité grâce à la commande FOC.

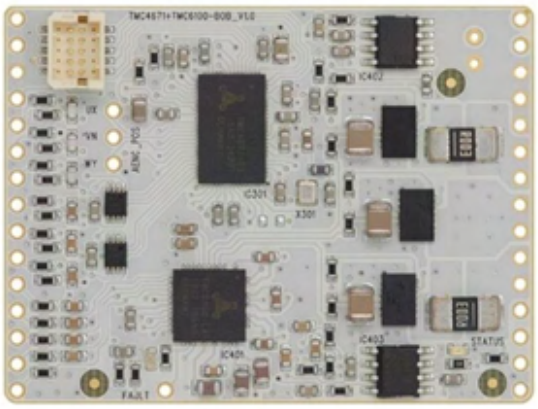


FIGURE 26 – Carte Trinamic TMC4671+TMC6100-BOB

Pour contrôler les moteurs avec la TMC4671+TMC6100-BOB, il faut d'une part envoyer les bons registres à la carte, et d'autre part configurer correctement la communication SPI entre l'ordinateur embarqué (ici une Rasberry PI3) et les cartes Trinamic. On peut voir les liaisons entre les trois cartes Trinamics et la Raspberry Pi3 sur la figure 27.

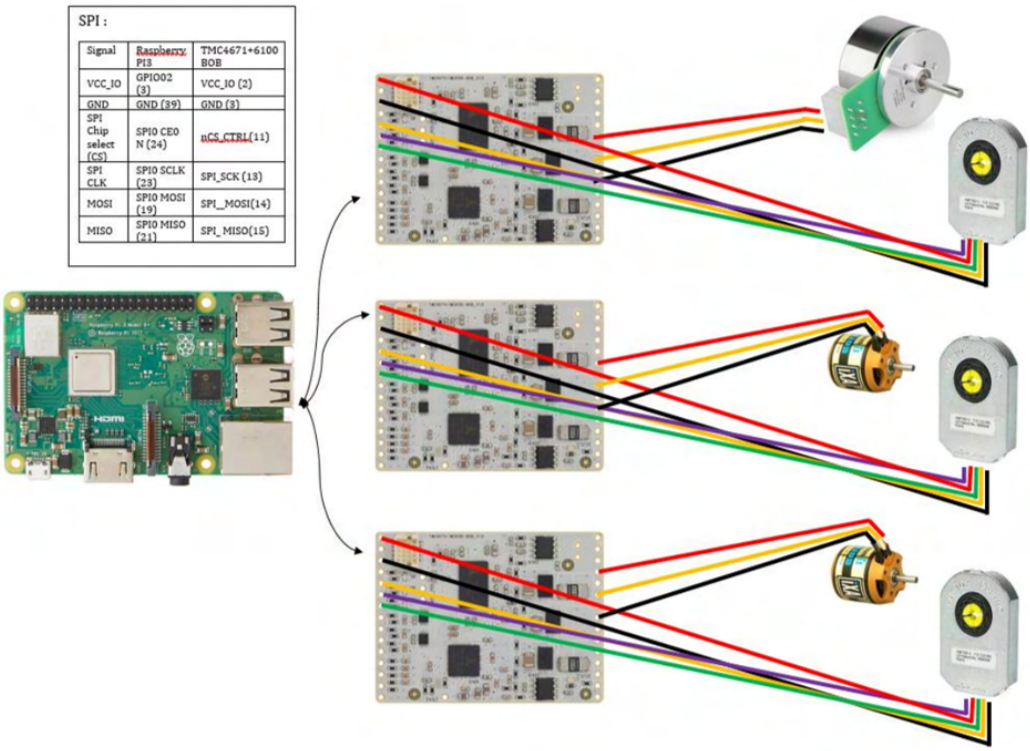


FIGURE 27 – Connectique de la TMC4671+TMC6100-BOB

Nous n'avons pas réussi à faire fonctionner le SPI, mais voici l'ensemble des pistes étudiées. Afin de paramétrer l'ensemble de notre carte (paramètres du moteur, des encodeurs etc.) il nous a fallu définir les valeurs de ses registres. Les valeurs que nous avons utilisées sont :

- Celles que nous avons exportées à partir du TMCL-IDE, avec la configuration que nous avons choisie en fonction de notre moteur (voir annexe 5)

- Celles que nous avons trouvées dans le Github de Trinamic [[8]] Il faut ensuite créer la liaison SPI entre la Raspberry PI3 et la Trinamic.

Les codes réalisant cette liaison SPI sont disponibles sur notre Github au lien suivant : `https://github.com/Katell-Lag/robot_nageur/blob/main/controle_robot/Trinamic/TMC4671%2BTMC6100-BOB/TMCAPI_EXAMPLE.zip`. Ils sont inspirés d'un tutoriel reliant une Raspberry Pi3 à une TMC5160-BOB [9]. Nous avons essayé d'adapter ce tutoriel de différentes manières pour faire fonctionner la liaison avec notre carte. Nous avons utilisé entre autres la librairie bcm2835 pour configurer le SPI. Les fonctions définies dans le code en langage C (voir figure 28) sont un exemple de test de protocole SPI que nous avons réalisé, plus précisément ce sont des fonctions chargées d'écrire dans les registres de la carte via SPI.

```c
#include <bcm2835.h>
#include "SPI_TMC.h"


// TMC4671 SPI wrapper
void tmc4671_writeDatagram(uint8 motor, uint8 address, uint8 x1, uint8 x2, uint8 x3, uint8 x4)
{
    int value = x1;
    value <<= 8;
    value |= x2;
    value <<= 8;
    value |= x3;
    value <<= 8;
    value |= x4;

    tmc40bit_writeInt(motor, address, value);
}

void tmc4671_writeInt(u8 motor, uint8 address, int value)
{
    tmc40bit_writeInt(motor, address, value);
}

int tmc4671_readInt(u8 motor, uint8 address)
{
    tmc40bit_readInt(motor, address);
    return tmc40bit_readInt(motor, address);
}

u16 tmc4671_readRegister16BitValue(u8 motor, u8 address, u8 channel)
{
    char tbuf[3], rbuf[3];
    u16 value;
    // clear write bit
    tbuf[0] = address & 0x7F;
    tbuf[1] = 0;
    tbuf[2] = 0;

    bcm2835_spi_transfernb (tbuf, rbuf, 3);

    value =rbuf[1];
    value <<= 8;
    value |= rbuf[2];

    return value;
}
```

FIGURE 28 – Extrait du code C pour la transmission SPI

Nous avons essayé de lire des données provenant de la carte mais sans succès. Il semble que la liaison ne se fasse pas. Cependant, après avoir réalisé des mesures via l'oscilloscope, nous avons pu observer que des trames étaient bien envoyées de la Raspberry Pi3 vers la Trinamic. Nous pensons donc que la Trinamic ne parvient pas à interpréter ces trames, puisque la Raspberry, elle, reçoit et envoie bien les trames. On remarque que lorsqu'on connecte matériellement les ports MOSI et MISO de la PI3 les données reçues et émises sont similaires : le SPI de la PI3 fonctionne (voir figure 29).

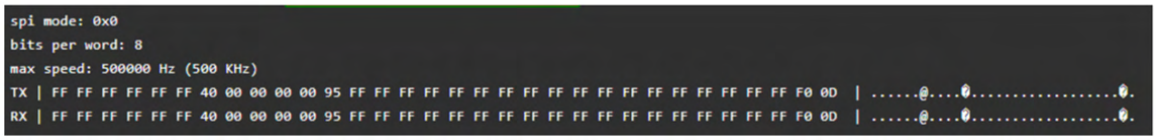


FIGURE 29 – Tests pour la transmission SPI de la Raspberry

On voit que les trames envoyées et reçues sont similaires. L'erreur de transmission provient donc bien de la carte Trinamic.

Nous avons pour finir essayé de connecter la carte BOB à la carte Landungsbrücke afin de vérifier si une communication « entre Trinamic » pourrait fonctionner mais sans résultat actuellement. Le montage que nous avons réalisé est celui présenté sur la figure 30.

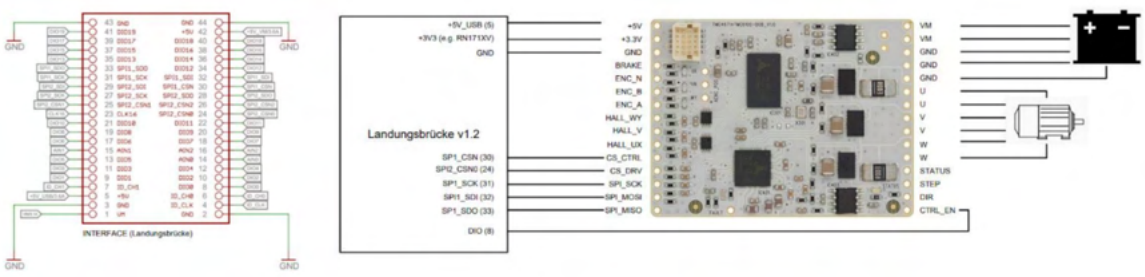


FIGURE 30 – Tentative de connexion entre cartes Trinamic

Finalement, toutes ces recherches nous ont permis d'approfondir notre compréhension de la transmission SPI, et également de réaliser de nombreux tests différents sur une nouvelle carte électronique et ainsi de pouvoir transmettre toutes nos découvertes aux chercheurs et étudiants qui se pencheront sur cette carte.

6 Conclusion

Ce stage fut une expérience très enrichissante, et l'occasion de mettre en pratique les connaissances apprises à l'ENSTA Bretagne. C'était également l'occasion de découvrir un autre laboratoire et ainsi de voir des méthodes et des matériels différents (par exemple la BeagleBone Blue).

Durant ce stage j'ai pu approfondir mes connaissances en mécanique, en informatique (notamment le langage C), en électronique et circuits électriques (avec le bouton d'arrêt d'urgence par exemple). J'ai aussi pu développer mes méthodes de recherches, définir un banc de tests et acquérir de l'expérience technique en termes de soudure, perçage, découpage etc.

J'ai pu appliquer mes connaissances techniques en robotique, en modélisation 3D et en programmation. J'ai aimé réaliser ce stage pour l'aspect robotique (commande de robot, simulation) et mécanique (modélisation, recherche de pièces ergonomiques). Entourées des professeurs de Polytech Montpellier, nous avons finalement pu mettre en place une architecture efficace en termes de résistance, poids et encombrement de la partie « épaule » de notre robot. Travailler avec une équipe comme celle-ci fut pour moi une très belle expérience.

Ce stage n'avait pas une mission bien précise à atteindre mais visait plutôt à développer au maximum le projet. Finalement, nous avons une simulation complète du robot, un contrôle fonctionnel en boucle ouverte (avec la BBBlue) et un premier prototype de l'épaule.

Références

[1] Anna AGOBIAN, Tatiana DEWILDEMA, Théo LEMAIRE, and Paul MASSON. Modélisation, conception et réalisation d'un robot nageur. 2020.

[2] Vincent Padois. Enchaînements dynamiques de tâches pour des manipulateurs mobiles à roues. aautomatique / robotique. institut national polytechnique de toulouse - inpt, 2005.français. <tel-00011677v2>.

[3] Mathias Samson. Étude expérimentale et numérique, en écoulement instationnaire, du trajet des bras en crawl à différentes allures de nage. thèse biomécanique et bio-ingénierie. poitiers : Université de poitiers, 2016.disponible sur internet <http ://theses.univ-poitiers.fr>.

[4] Alain Liégeois. Modélisation et commande des robots manipulateurs, réf : R7730 v1. 1998.

[5] Klemens Konhäuser. Driving a linear stage with the tmc4671 – firmware adaptation, consulté le 15 juillet 2021. `http://blog.trinamic.com/2020/02/28/driving-a-linear-stage-with-the-tmc4671-firmware-adaptation/`, 2020.

[6] Librairie pytrinamic, consultée en juillet 2021. `https://github.com/trinamic/PyTrinamic`.

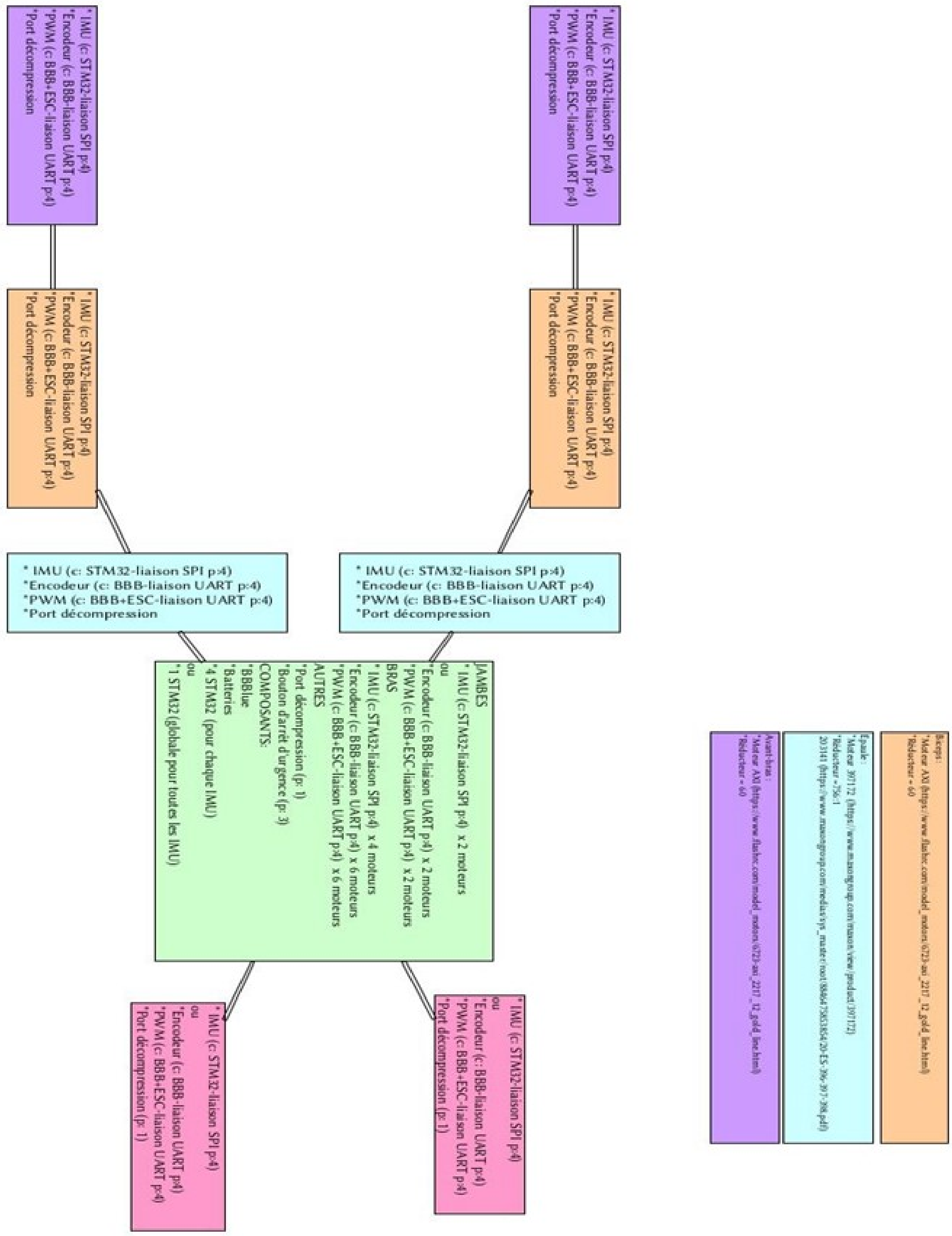
[7] Contrôle en boucle ouverte, en python (consulté le 20 juillet 2021). `https://github.com/trinamic/PyTrinamic/blob/master/PyTrinamic/examples/evalboards/TMC4671/TMC4671_eval_BLDC_open_loop.py`.

[8] Github de trinamic, consulté en juillet 2021. `https://github.com/trinamic`.

[9] Mario Nolten. Driving stepper motors with the new tmc5160 by using trinamic's api on a raspberry pi, consulté le 22 juillet 2021. `http://blog.trinamic.com/2018/02/19/stepper-motor-with-tmc5160/`.

Annexes

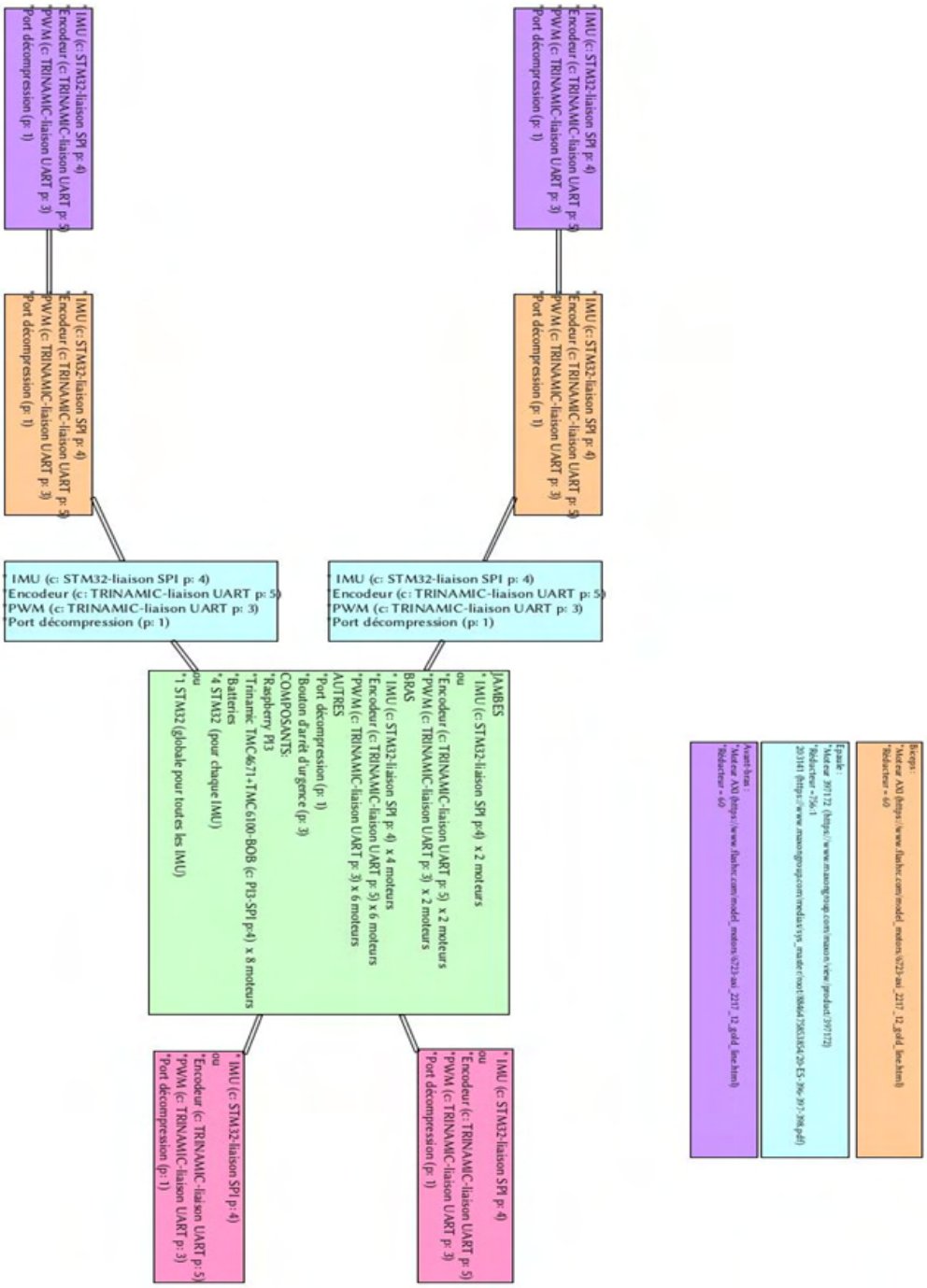
Annexe 1 : Architecture électronique du bras avec la BBlue



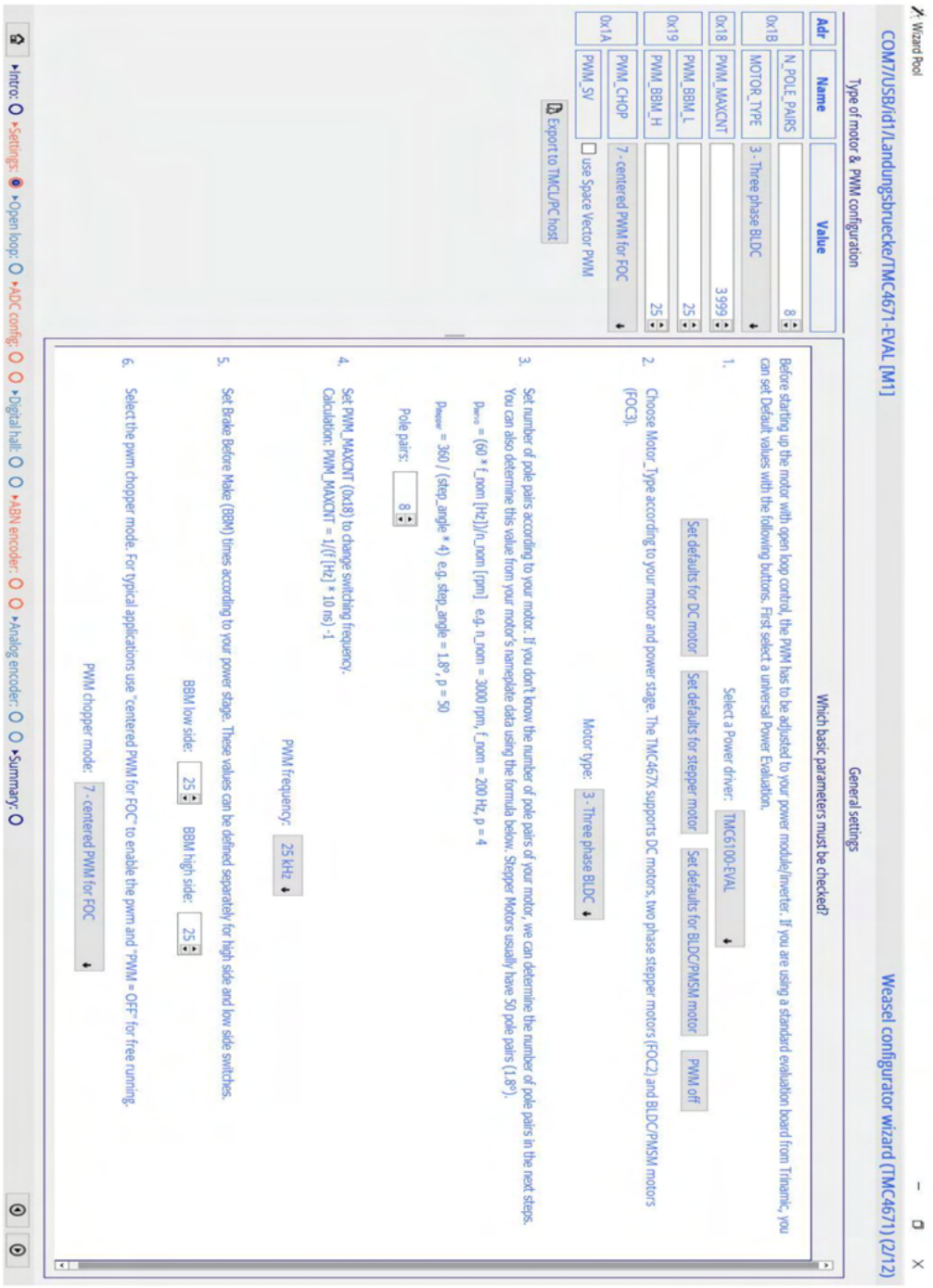
Annexe 2 : Code C de la commande des moteurs par la BBlue

```c
// Main loop runs at frequency_hz
while(running){
    switch(mode){
    case SWEEP:
        //moteur epaule: thr
        //moteur 1 = moteur du bas, ch = 1, thr2
        //moteur 2 = moteur du haut, ch = 2, thr3
        // increase or decrease position each loop
        // scale with frequency
        thr += dir * sweep_limit / frequency_hz;

        // reset pulse width at end of sweep
        if(thr > sweep_limit){
            thr = sweep_limit;
            dir = -1;
        }

        else if(thr < 0){
            thr = 0;
            dir = -1;
        }
        // send result
        rc_servo_send_esc_pulse_normalized(ch,thr); //moteur epaule lancé

        // MACHINE D'ETAT DES DEUX MOTEURS BRAS ET AVANT-BRAS

        if (indice == 0 && rc_encoder_read(4) <= -800000){  //indice 0: tour du bras de 180 degrés
            printf("CHANGEMENT\n");
                indice = 1;
        }
        if (indice == 1 && rc_encoder_read(1) < -220000){   //indice 1: les deux moteurs tournent pour degré de liberté bras
            indice = 2;
        }
        if (indice == 2 && rc_encoder_read(1) < -480000){   //indice 2: moteur 1 tourne pour plier avant-bras
            indice = 3;
        }

        if (indice == 3 && rc_encoder_read(4) <= -1400000){ //indice 3: attente, seul le moteur epaule tourne
            indice = 4;
        }
        if (indice == 4 && rc_encoder_read(1) > -240000){   //indice 4: moteur 1 à l'envers, dépliement de l'avant bras
            indice = 5;
        }
        if (indice == 5 && rc_encoder_read(1) < -450000){   //indice 5: les deux moteurs tournent pour finir rotation bras
            indice = 6;
        }

        //indice 6: les 2 moteurs sont à l'arrêt
        if (indice == 1 || indice == 2 || indice == 5){

            thr2 += dir2*sweep_limit / frequency_hz;
            if (thr2 > sweep_limit){
                thr2 = sweep_limit;
                dir2 = -1;
            }
            else if (thr2<0){
                thr2 = 0;
                dir2 = -1;
            }

            rc_servo_send_esc_pulse_normalized(1, thr2); //moteur bras
        }

        if (indice == 6){
            rc_servo_send_esc_pulse_normalized(1, 1-thr2);
            printf("MARCHE ARRIERE TOUTE\n");
        }

        //COMMANDE MOTEUR AVANT BRAS
        if (indice == 1 || indice == 5){
            thr3 +=dir3*sweep_limit /frequency_hz;
            if (thr3 > sweep_limit){
                thr3 = sweep_limit;
                dir3 = -1;
                }

            else if (thr3 < 0){
                thr3 = 0;
                dir3 = -1;
            }

            printf("THR3= %lf\n",1-thr3);
            rc_servo_send_esc_pulse_normalized(2,1- thr3); //moteur avant bras
        }

        if (indice == 2 || indice == 3 || indice == 4 || indice == 6){ //ARRET MOTEUR 2
            printf("ARRET MOTEUR 2\n");
            rc_servo_send_esc_pulse_normalized(2, 0.5);
        }
        if (indice == 3 || indice == 6){ //ARRET MOTEUR1
            printf("ARRET MOTEUR 1\n");
            rc_servo_send_esc_pulse_normalized(1, 0.5);
        }

        if (compteur %1 == 0){
        rc_usleep(1000);

        fprintf(fichier1, "%10d\n", rc_encoder_read(4));
        fprintf(fichier2, "%10d\n", rc_encoder_read(1));
        fprintf(fichier3, "%10d\n", rc_encoder_read(2));
        rc_usleep(1000);
        }
        compteur +=1;

        if  ((rc_encoder_read(4)<=-2100000)) //ARRET MOTEUR EPAULE APRES UN TOUR COMPLET
        {
            rc_servo_send_esc_pulse_normalized(ch, 0.5);
            return -1;
        }

        break;
```

Annexe 3 : Architecture électronique du bras avec la Trinamic



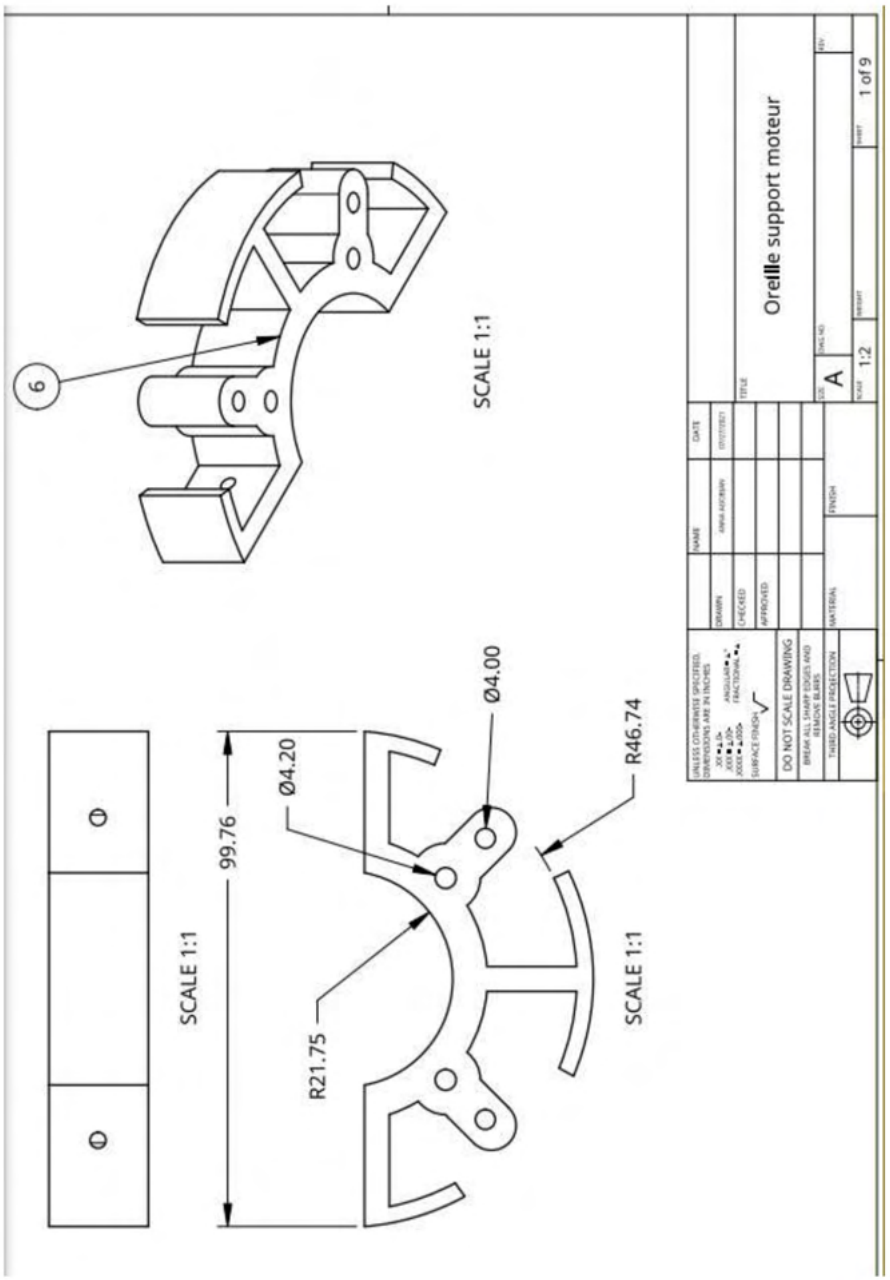
Annexe 4 : Définition des paramètres du moteur dans le TMCL-IDE

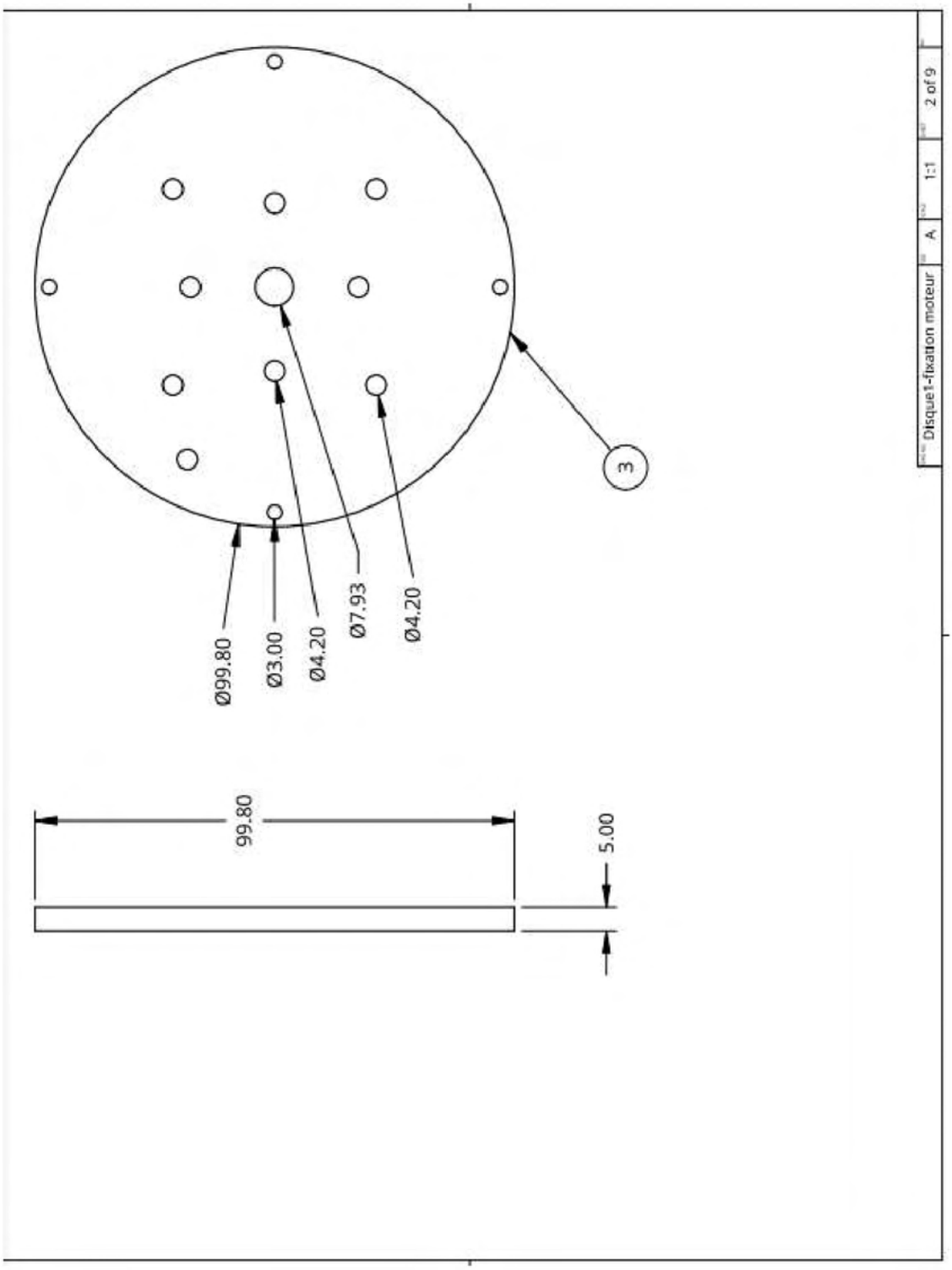


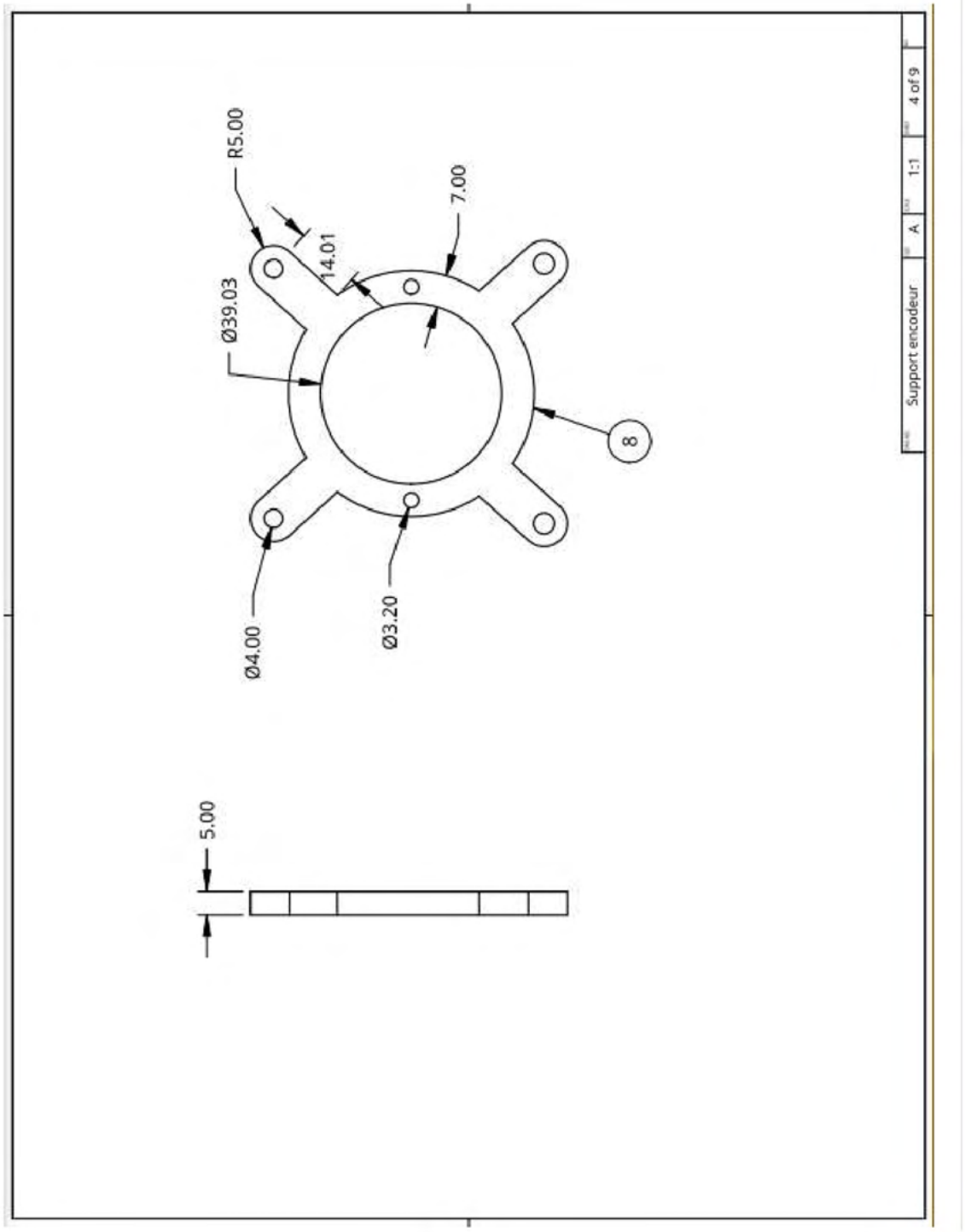
Annexe 5 : Configuration des registres de la Trinamic pour faire tourner le moteur

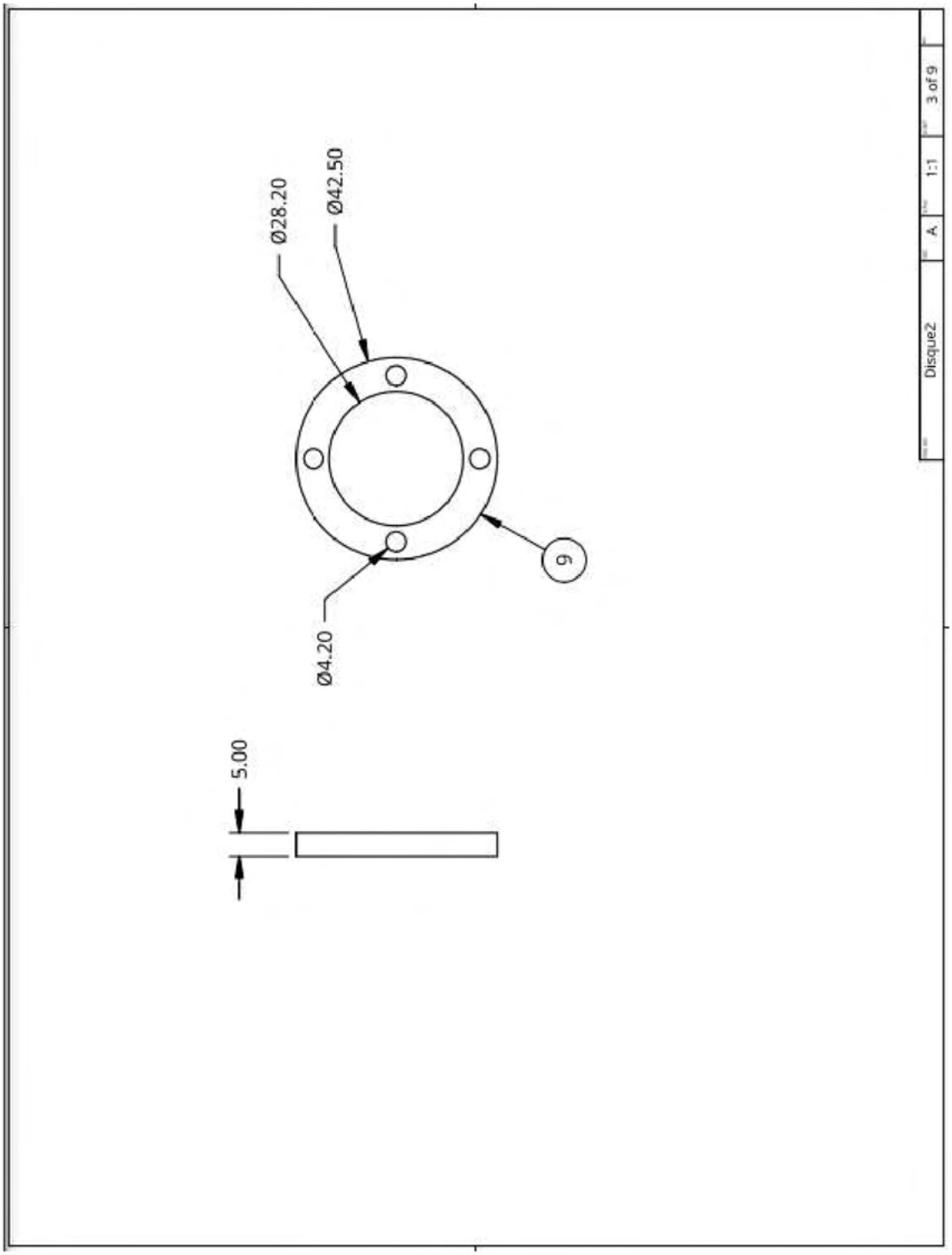
```
#include "hal/ic/TMC4671.h"
// Motor type &  PWM configuration
tmc4671_writeInt(0, TMC4671_MOTOR_TYPE_N_POLE_PAIRS, 0x00030008);
tmc4671_writeInt(0, TMC4671_PWM_POLARITIES, 0x00000000);
tmc4671_writeInt(0, TMC4671_PWM_MAXCNT, 0x00000F9F);
tmc4671_writeInt(0, TMC4671_PWM_BBM_H_BBM_L, 0x00001919);
tmc4671_writeInt(0, TMC4671_PWM_SV_CHOP, 0x00000007);

// ADC configuration
tmc4671_writeInt(0, TMC4671_ADC_I_SELECT, 0x09000100);
tmc4671_writeInt(0, TMC4671_dsADC_MCFG_B_MCFG_A, 0x00100010);
tmc4671_writeInt(0, TMC4671_dsADC_MCLK_A, 0x20000000);
tmc4671_writeInt(0, TMC4671_dsADC_MCLK_B, 0x00000000);
tmc4671_writeInt(0, TMC4671_dsADC_MDEC_B_MDEC_A, 0x014E014E);
tmc4671_writeInt(0, TMC4671_ADC_I0_SCALE_OFFSET, 0x01008001);
tmc4671_writeInt(0, TMC4671_ADC_I1_SCALE_OFFSET, 0x01008001);

// Open loop settings
tmc4671_writeInt(0, TMC4671_OPENLOOP_MODE, 0x00000000);
tmc4671_writeInt(0, TMC4671_OPENLOOP_ACCELERATION, 0x0000003C);
tmc4671_writeInt(0, TMC4671_OPENLOOP_VELOCITY_TARGET, 0xFFFFFFF6);


// Feedback selection
tmc4671_writeInt(0, TMC4671_PHI_E_SELECTION, 0x00000002);
tmc4671_writeInt(0, TMC4671_UQ_UD_EXT, 0x00000A60);
// ===== Open loop test drive =====
// Switch to open loop velocity mode
tmc4671_writeInt(0, TMC4671_MODE_RAMP_MODE_MOTION, 0x00000008);
// Rotate right
tmc4671_writeInt(0, TMC4671_OPENLOOP_VELOCITY_TARGET, 0x0000003C);
wait(2000);


// Rotate left
tmc4671_writeInt(0, TMC4671_OPENLOOP_VELOCITY_TARGET, 0xFFFFFFC4);
wait(4000);
// Stop
tmc4671_writeInt(0, TMC4671_OPENLOOP_VELOCITY_TARGET, 0x00000000);
wait(2000);
tmc4671_writeInt(0, TMC4671_UQ_UD_EXT, 0x00000000);
```

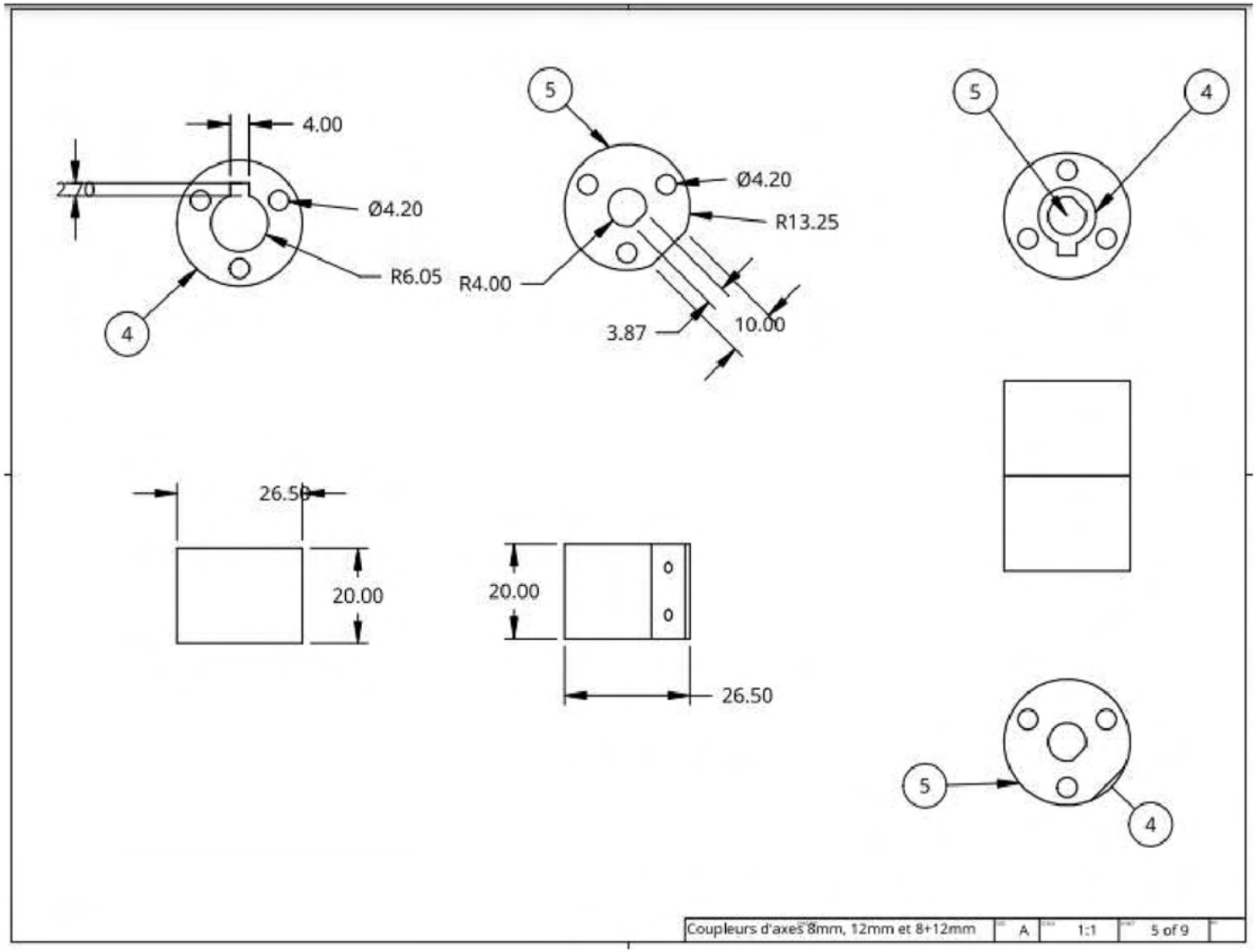
Annexe 6 : Mises en plan des pièces réalisées avec le logiciel OnShape

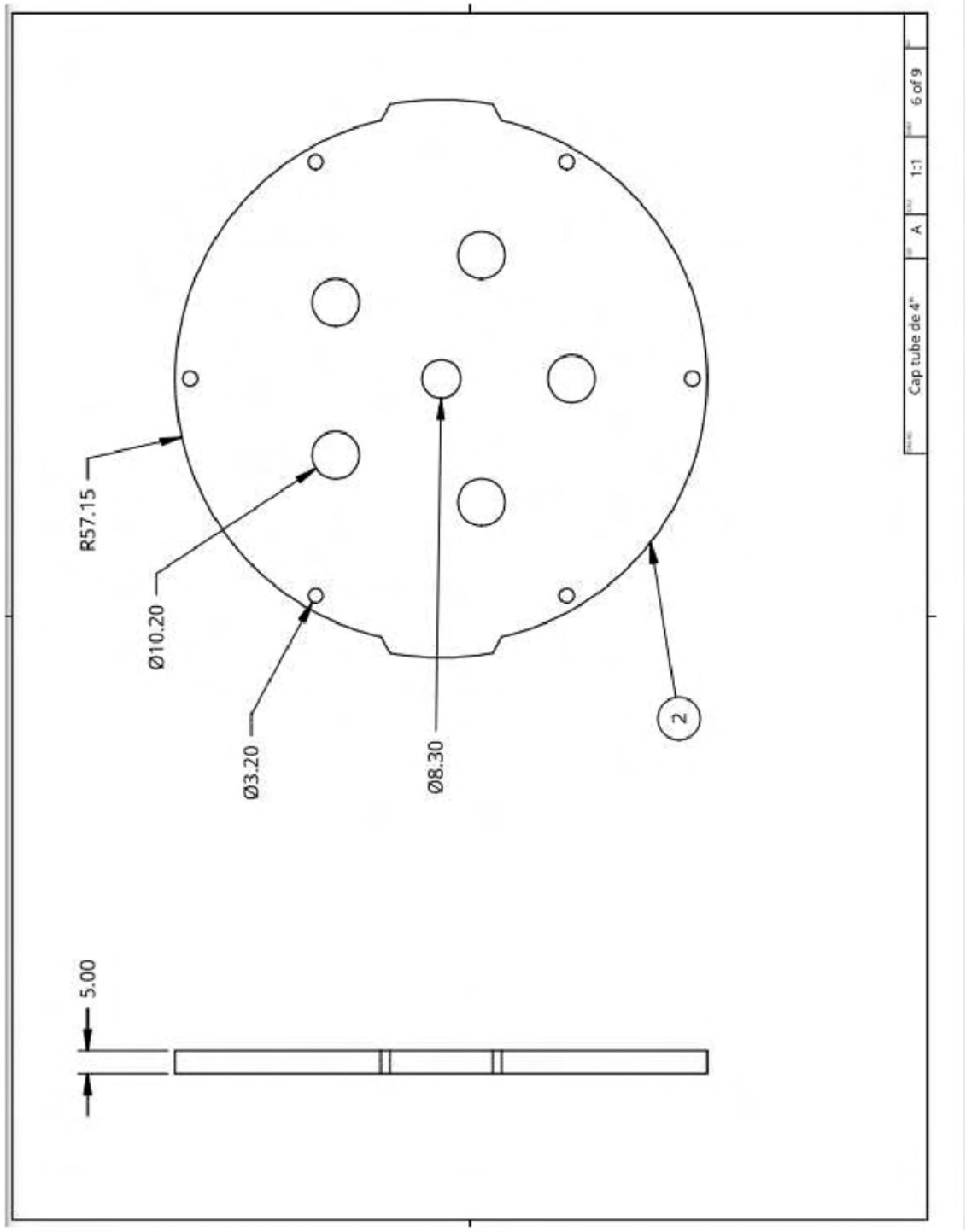


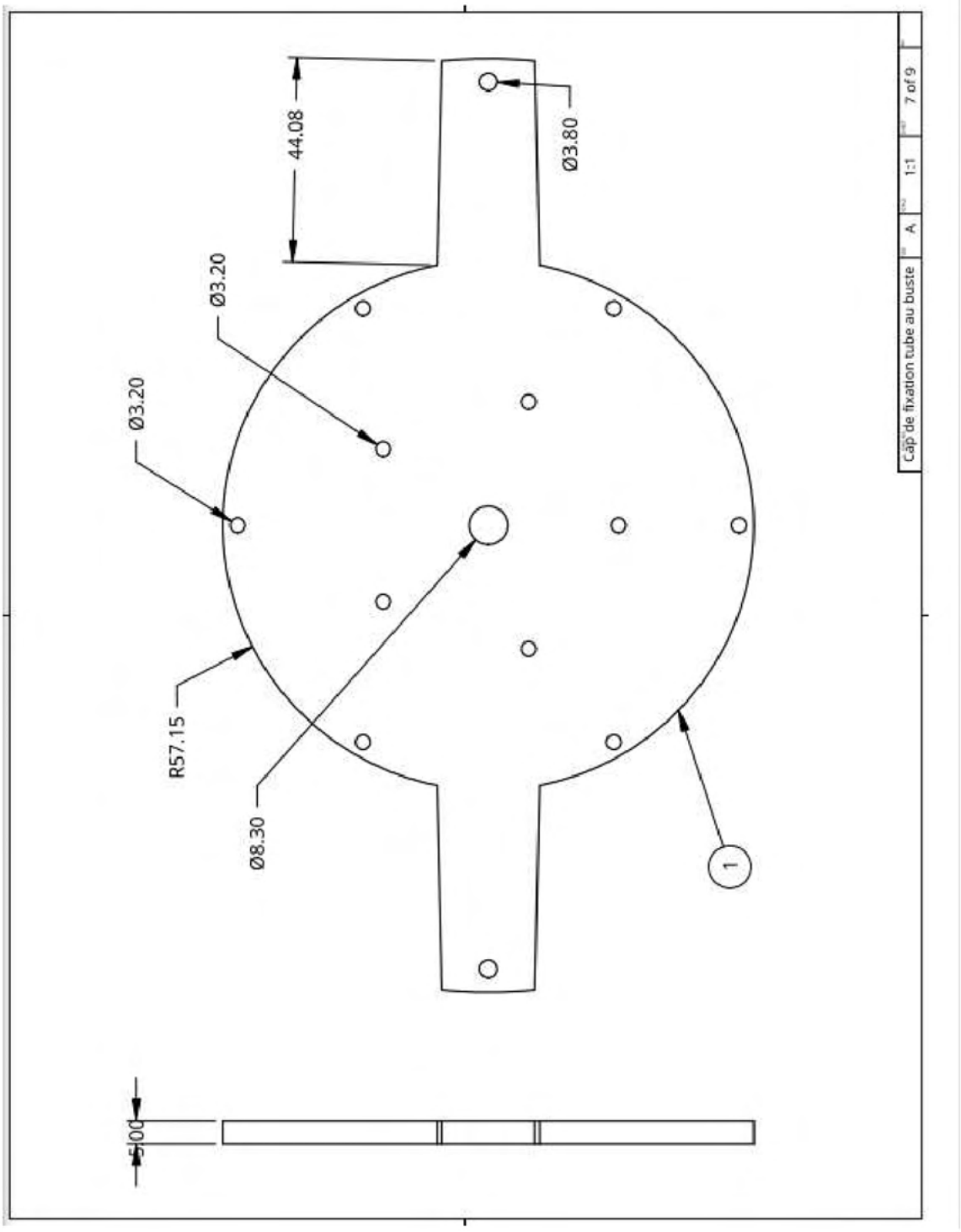

Disque1-fixation moteur
A
1:1
2 of 9
3
Ø99.80
Ø3.00
Ø4.20
Ø7.93
Ø4.20
99.80
5.00

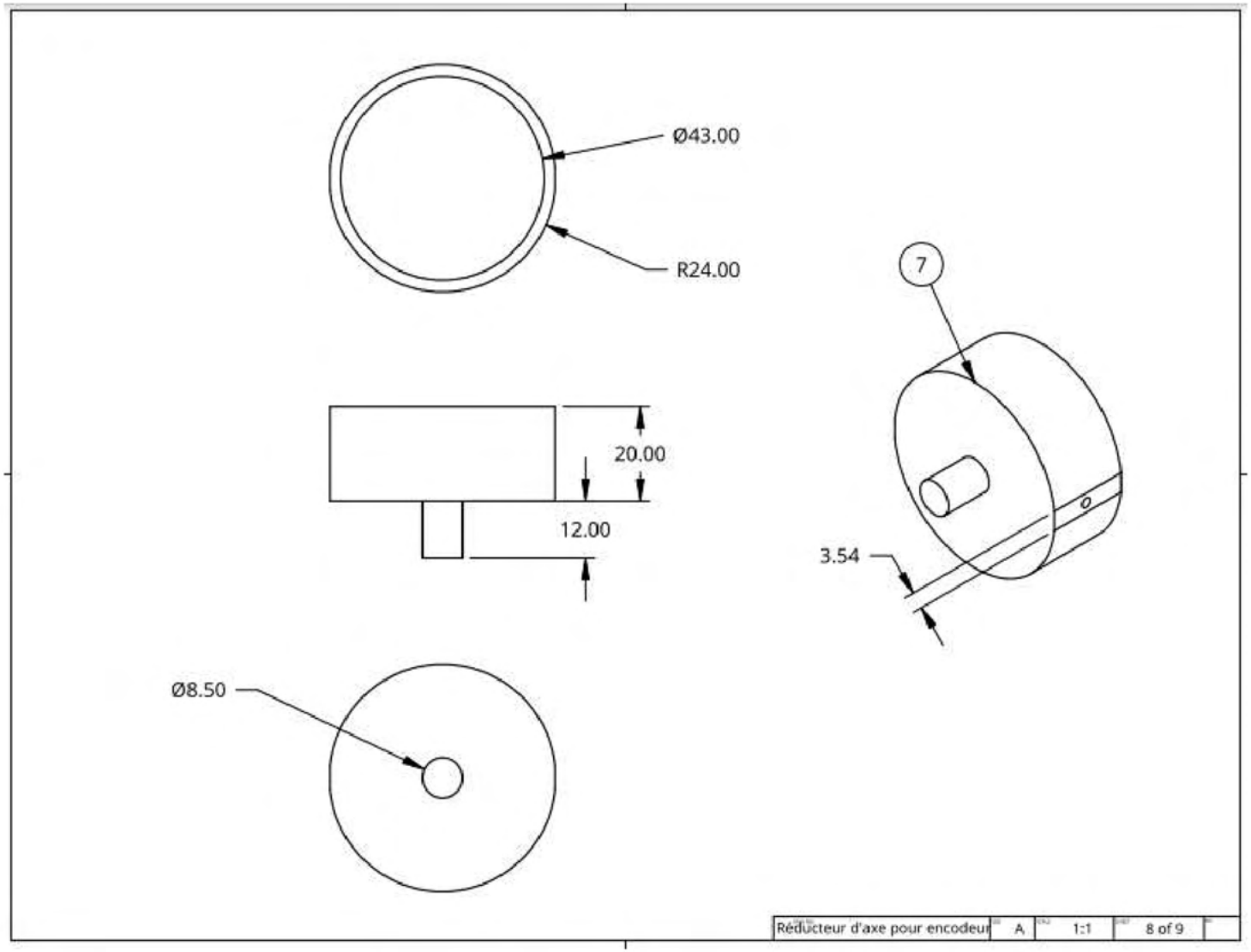

R5.00
14.01
7.00
Ø39.03
8
Ø4.00
Ø3.20
5.00
Support encodeur
A
1:1
4 of 9

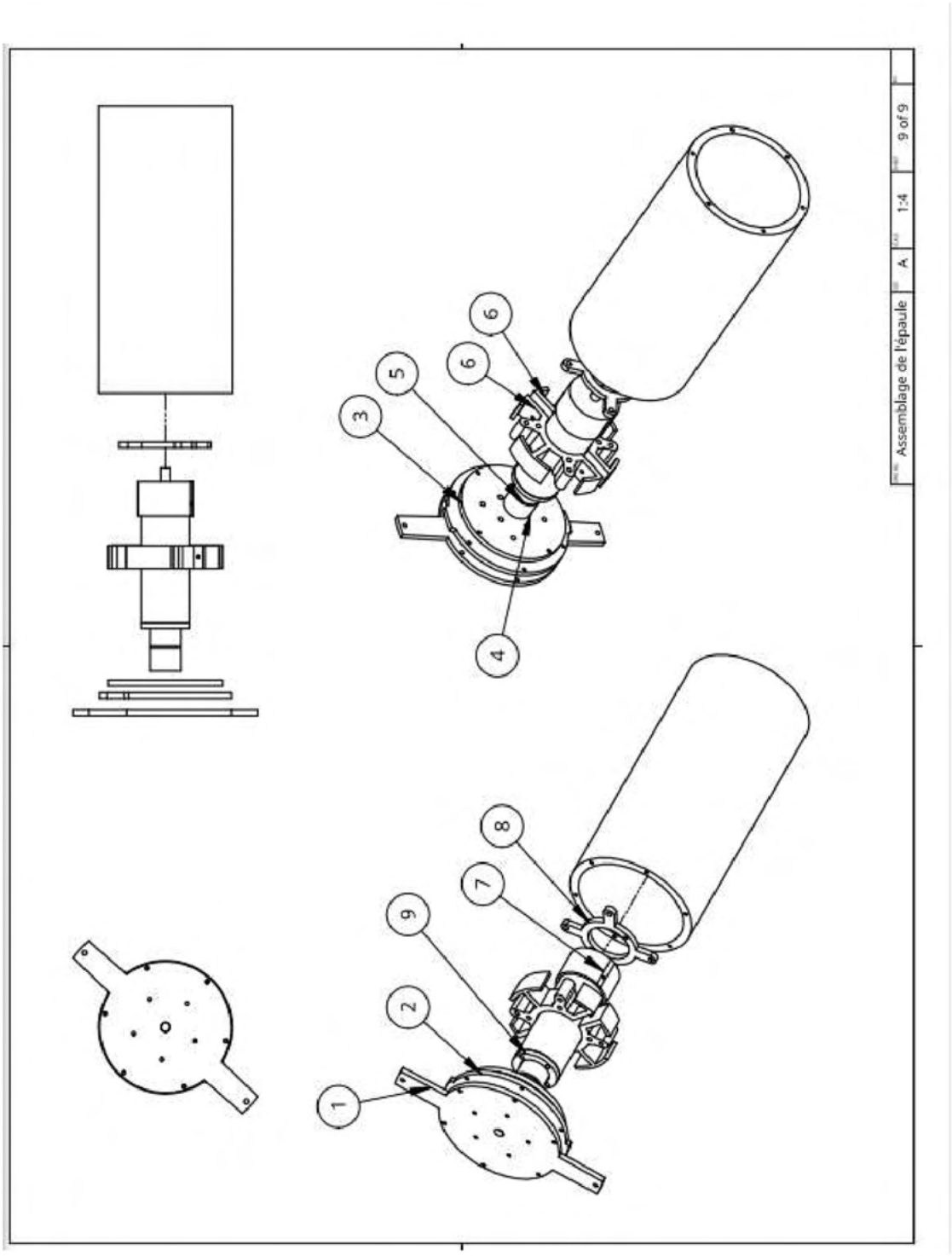

Ø28.20
Ø42.50
Ø4.20
6
5.00
Disque2
A
1:1
3 of 9


4.00
2.70
Ø4.20
R6.05
4
5
Ø4.20
R13.25
R4.00
3.87
10.00
5
4
26.50
20.00
20.00
26.50
5
4
Coupleurs d'axes 8mm, 12mm et 8+12mm
A
1:1
5 of 9


R57.15
Ø10.20
Ø3.20
Ø8.30
2
5.00
Cap tube de 4"
A
1:1
6 of 9


44.08
Ø3.80
Ø3.20
Ø3.20
R57.15
Ø8.30
1
5.00
Cap de fixation tube au buste
A
1:1
7 of 9


Ø43.00
R24.00
20.00
12.00
7
3.54
Ø8.50
Réducteur d'axe pour encodeur
A
1:1
8 of 9


Assemblage de l'épaule
A
1:4
9 of 9
1
2
3
4
5
6
6
7
8
9