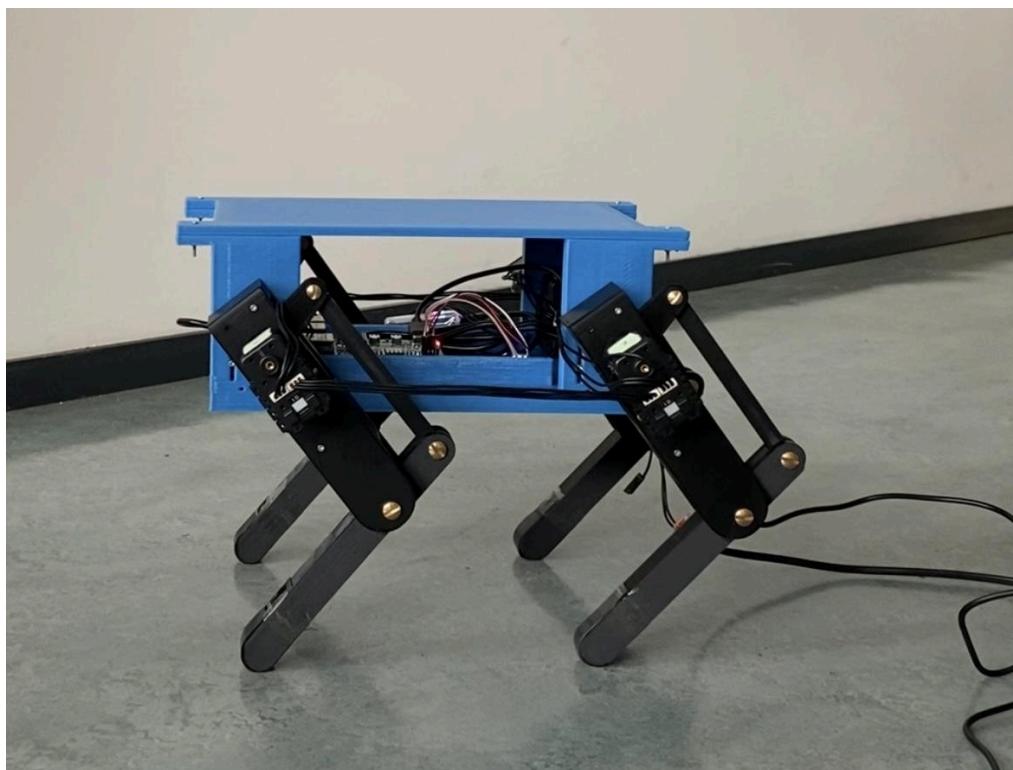


# INSAdogo, un robot quadrupède dopé à l'IA



**Équipe-projet 5A:**

LEBEAU Damien

TOURE Mouhamadou

GOMES Killian

PHOUTTHASAK Anousith

TRAN Jonathan

**Tuteur enseignant :**

DELPRAT Sébastien

**Codification :**

PLP23INT23

**Année :**

2023/2024

# Sommaire

<b>Sommaire.....</b>	<b>2</b>
<b>État de l'art.....</b>	<b>3</b>
<b>Cahier des charges.....</b>	<b>5</b>
Contexte.....	5
Objectif.....	5
Périmètre.....	5
Descriptions fonctionnelles des besoins.....	6
Enveloppement budgétaire.....	6
Délais.....	6
<b>Analyse du projet.....</b>	<b>7</b>
Acteurs du projet et clients.....	7
Diagramme de méthodologie et fiche de tâche.....	7
<b>Avancement du projet.....</b>	<b>8</b>
Choix des composants.....	8
Conception du robot.....	12
Modélisation 3D.....	12
Étude de déformation.....	13
Étude de déplacements.....	14
Étude de contraintes.....	15
Impression 3D.....	16
Configuration de la Raspberry Pi 4.....	16
Installation de l'OS et configuration.....	16
Mise en place de la communication wifi.....	18
Création du serveur virtuel et connexion avec la Raspberry.....	18
Début du prototypage du robot.....	19
Prototypage du robot.....	19
Principe du cycle de marche.....	23
Programmation.....	25
Vérification et configuration des servomoteurs.....	25
Rédaction du cycle de marche.....	26
Une seule patte.....	26
Deux pattes.....	27
Quatres pattes.....	27
En cas de chute.....	28
Proposition d'un programme basée sur l'IA.....	29
Prototype final du robot.....	31
<b>Conclusion.....</b>	<b>32</b>
<b>Bibliographie.....</b>	<b>33</b>

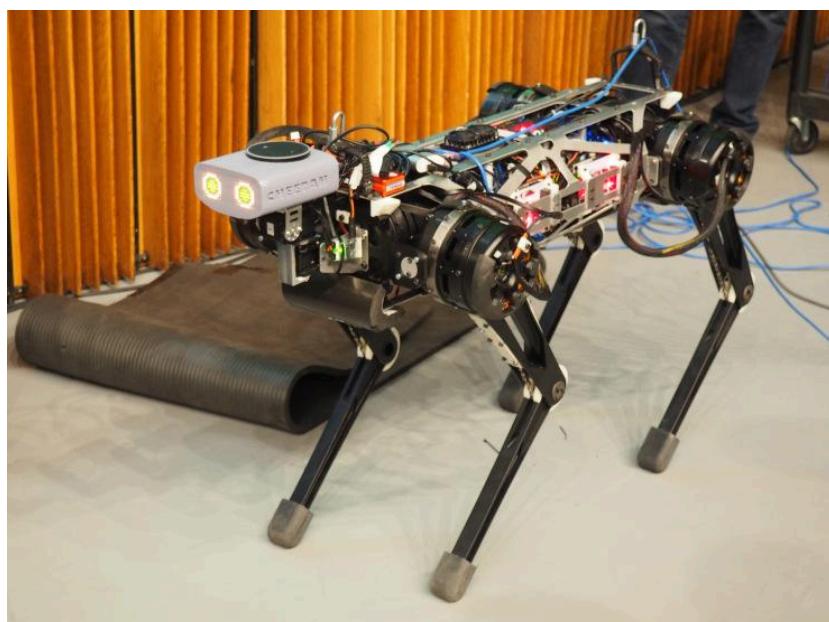
## État de l'art

Avant de commencer notre projet, nous allons réaliser une synthèse sur l'ensemble des travaux et solutions connus dans le domaine des robots quadrupèdes qui utilisent l'apprentissage par renforcement pour se déplacer. Nous allons donc faire un état de l'art du sujet avec une liste de robots existants sur le marché :

- Cheetah 3 :

Cheetah 3 est un robot quadrupède développé par le Massachusetts Institute of Technology (MIT) qui utilise l'apprentissage par renforcement pour améliorer son équilibre et sa locomotion.

Les chercheurs ont utilisé des algorithmes d'apprentissage par renforcement basés sur des modèles pour permettre au robot d'apprendre à se déplacer efficacement dans des environnements inconnus et difficiles. [1]



- ANYmal :

ANYmal est un robot développé par la société suisse ANYbotics, conçu pour opérer dans des environnements complexes et hostiles.

L'apprentissage par renforcement est utilisé pour permettre à ce robot de s'adapter à des terrains variés, de surmonter des obstacles et d'optimiser sa locomotion. [2]



- Laikago :

Laikago est un robot créé par Unitree Robotics. Il utilise des algorithmes d'apprentissage par renforcement pour améliorer sa stabilité et son agilité.

L'approche d'apprentissage par renforcement permet au robot de s'ajuster en temps réel aux changements de l'environnement, ce qui est particulièrement utile dans des situations imprévues. [3]



- HyQReal :

HyQReal est un robot développé par l'Istituto Italiano di Tecnologia (IIT) en Italie, conçu pour des applications de recherche et de sauvetage.

L'apprentissage par renforcement est utilisé pour optimiser la locomotion du robot dans des environnements difficiles, en lui permettant de s'adapter à des surfaces variées et de prendre des décisions intelligentes lors de la navigation. [4]



## Cahier des charges

### Contexte

Pour diversifier et améliorer les différents enseignements, M.DELPRAT, automatien et enseignant à l'INSA Hauts-de-France, souhaite que notre équipe produise un robot-chien qui sera étudié et amélioré au fil des années.

### Objectif

Concevoir un robot quadrupède simple et facile d'utilisation et de compréhension permettant une reprise facile du projet. L'idée est d'utiliser les ressources déjà disponibles sur internet pour accélérer le processus de développement étant donné les conditions de développement non optimales.

### Périmètre

Ce robot sera destiné aux futurs étudiants de l'INSA dans un but pédagogique afin qu'ils puissent continuer à y travailler dessus pour un futur plateau-projet, ou soit qu'ils puissent le découvrir et l'étudier lors de Travaux Pratiques (TP) ou autres projets.

## Descriptions fonctionnelles des besoins

Dans le cadre de notre plateau-projet (PLP) de fin d'études, le cahier de charge convenu avec le client est :

- Conception d'un robot quadrupède d'une taille moyenne de 20 cm
- Commandable par Wi-Fi
- Équipé de 8 servomoteurs
- Capable d'effectuer un cycle de marche en ligne droite
- Capable de se relever en cas de chute
- Proposition d'un programme basé sur l'apprentissage par renforcement (IA)

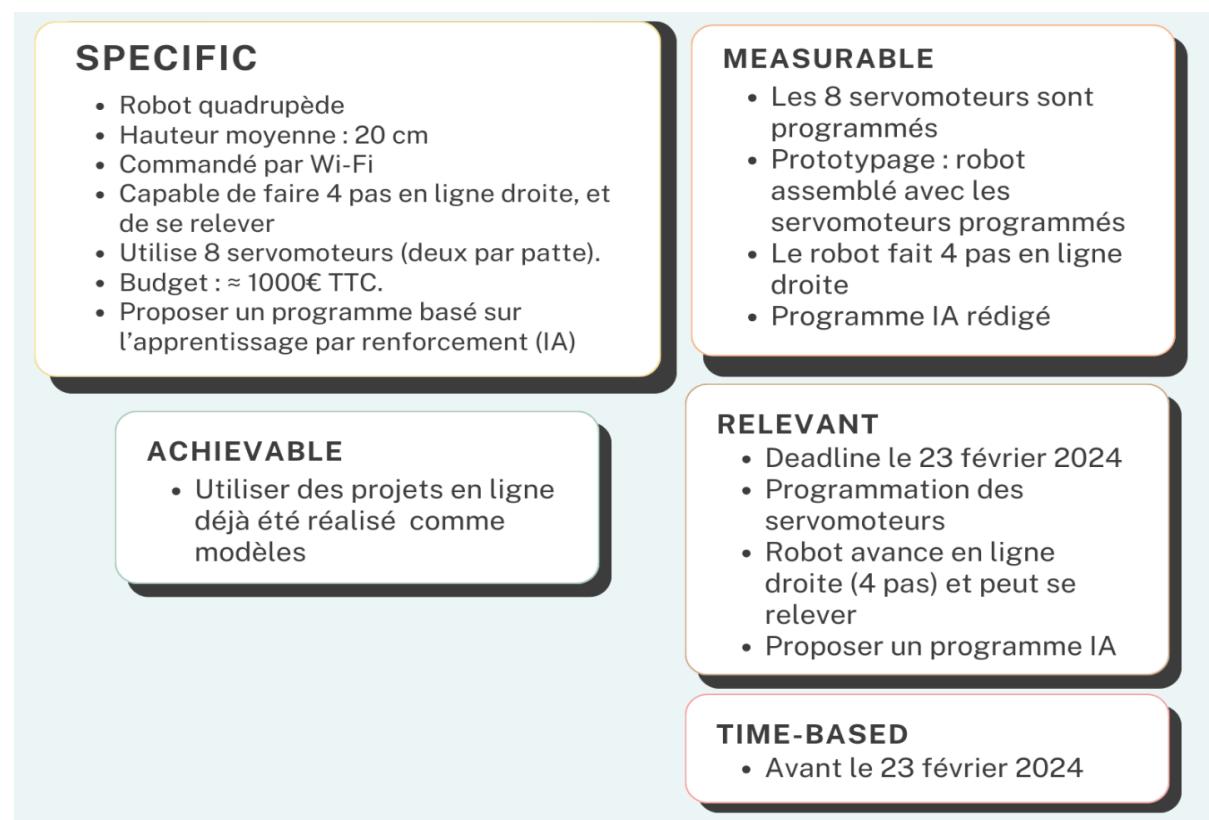
## Enveloppement budgétaire

Le budget total alloué à ce projet est 1000€ TTC.

## Délais

La date limite de ce projet est le 23 février 2024.

Voici un résumé visuel via la méthode SMART qui seront nos repères essentiels et qui guideront notre parcours vers la réalisation du projet INSAdogo.



# Analyse du projet

## Acteurs du projet et clients

Pour le bon déroulement de notre projet, il est nécessaire d'impliquer des acteurs autres que ceux de notre équipe-projet.

Nous avons d'abord les fournisseurs de composants, chez qui nous commandons les composants nécessaires au bon fonctionnement de notre robot.

Ensuite, après commande des composants, le département d'électronique se charge de financer l'achat de nos composants.

Le LAMIH et le service d'impression de l'INSA FABLAB s'occupent d'imprimer les pièces du robot en 3D.

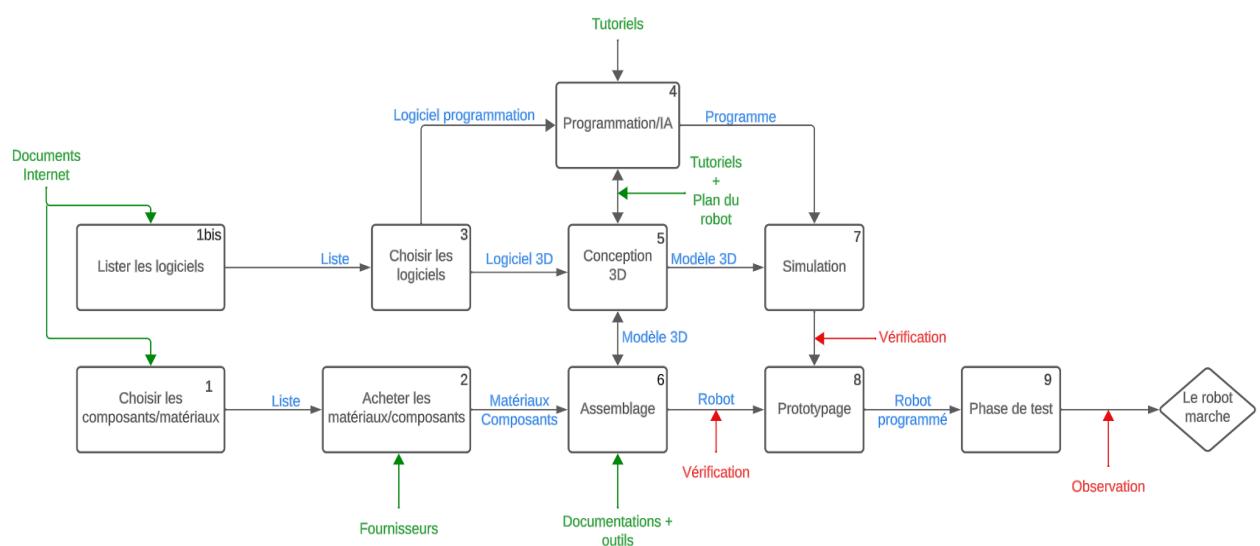
En termes de clients (à qui sera destiné notre robot), nous avons : notre tuteur M.DELPRAT et les étudiants de l'INSA.

M.DELPRAT est notre client intermédiaire. En effet, après la fin du plateau-projet, il récupérera notre robot.

Enfin, les étudiants de l'INSA sont notre client final. Comme précédemment dit dans le cahier des charges, ils pourront étudier et travailler sur le robot lors de TP ou projets.

## Diagramme de méthodologie et fiche de tâche

Dans le cadre de notre projet de conception d'un robot quadrupède commandé par Wi-Fi, il est essentiel d'établir une méthodologie claire et organisée pour assurer son succès. Cette méthodologie, représentée par un diagramme détaillant les différentes étapes du processus, ainsi qu'une fiche de tâches assignant les responsabilités des différents membres de l'équipe



Voici un exemple d'une fiche de tâche :

N° de tâche : 5	Nom de tâche : Conception 3D
- Responsable execution : K. GOMES	
- Responsable reporting : A. PHOUTTHASAK	
- Personne informée : D. LEBEAU	
Date de début : 01/11/2023	
Date de fin / durée : 29/11/2023	
Liste des entrées :	
- Logiciel conception 3D	
- Plan du robot avec dimension	
- Tutoriel(s)	
Liste des sorties :	
- Modèle de robot chien 3D	
- Rapport d'étude de mouvement	
Description de la tâche :	
- Réalisation du modèle 3D du robot chien en adéquation avec les plans réalisés au préalable (emplacement pour la carte électronique, les servomoteurs, etc.).	
- Vérification du modèle avec la personne à informer.	
- Etude de mouvement du modèle	
Niveau de performance/qualité à atteindre sur les sortie et Méthode de validation des sorties :	
- Géométrie respectée (forme, dimension)	
- Emplacement dédié aux composants	
- Validé grâce à la simulation sur l'interface graphique retenue	
Ressources nécessaires / Budget :	
Logiciel CAO / Budget : 0€ (si licence universitaire)	
Plan(s) du robot / Budget : 0€	
Documentation associée	
Bibliographie	

## Avancement du projet

### Choix des composants

Comme indiqué dans le cahier des charges, ce robot sera destiné aux futurs étudiants de l'INSA dans un but pédagogique afin qu'ils puissent continuer à travailler dessus pour un futur plateau-projet. C'est ainsi que nous avons choisi d'utiliser une Raspberry Pi 4 model b 4GB pour permettre aux futurs utilisateurs de pouvoir mettre en place directement des fonctions beaucoup plus avancées à savoir l'intégration de l'intelligence artificielle par exemple.



Raspberry Pi 4 Model B

Ci dessous, nous avons listé les avantages et inconvénients de notre carte:

- Avantages :

- Wifi bi-bande intégré, Bluetooth 5, GigaBit Ethernet
- 2 ports USB 3.0
- 2 ports micro HDMI en sortie vidéo
- 40 broches GPIO, utile pour ajouter et améliorer le prototype au fur et à mesure
- Bonne puissance de calcul local (bien pour le traitement d'image dans une optique d'amélioration)
- Stockage possible avec des micro SD, disques durs ou clés USB

- Inconvénients :

- Consommation d'énergie
- Pas de broche analogique, il faudra acheter des convertisseurs externes ADC

Concernant le choix des servomoteurs, nous sommes partis sur des servomoteurs compatibles avec notre Raspberry Pi 4 afin d'éviter d'acheter des composants supplémentaires. De plus, notre servomoteur doit avoir une rétroaction en position. Cette donnée sera utile lors des futurs PLP pour l'implémentation de l'IA.

Voici une étude comparative de différents servomoteurs qui pourraient être adaptés à notre application :

Servomoteurs	Données techniques	Caractéristiques
<b>MG996R(version numérique)</b>	<p>Couple max: 11 kg.cm= 1,08 N·m</p> <p>Operating speed: 0.17 s/60°</p> <p>Operating Voltage is +5V typically</p> <p>Current: 500-900mA (6V), Rotation : 0°-120°</p> <p>Weight of motor : 55g</p> <p>Dimension : 40,7 mm x 19,7 mm x 42,9 mm</p>	<ul style="list-style-type: none"> <li>• Bon rapport qualité-prix,</li> <li>• Largement utilisé dans les projets de robotique,</li> <li>• Disponible en version numérique (MG996R), compatible avec la Raspberry Pi</li> <li>• <b>Pas de rétroaction en position</b></li> </ul>
<b>Dynamixel AX-12A</b>	<p>Couple max: 1.5N.m</p> <p>Operating speed: 0.114s/60°</p> <p>Operating Voltage is 12V</p> <p>Current: 600mA à 900mA</p> <p>Rotation : 0°- 300°</p> <p>Weight of motor : 54g</p> <p>Dimension : 32x50x40 mm</p>	<ul style="list-style-type: none"> <li>• Précision élevée</li> <li>• Communication série bidirectionnelle</li> <li>• Capable de mouvements complexes</li> <li>• Compatible avec la Raspberry Pi</li> <li>• Rétroaction en position</li> </ul>
<b>TowerPro MG995</b>	<p>Couple max: 13 kg/cm</p> <p>Operating speed: 0.13 à 0.17s/60°</p> <p>Operating Voltage : 3V à 7.2V</p> <p>Current: 110 mA à 1450 mA</p> <p>Rotation : 0°-120°</p> <p>Weight of motor : 65g</p> <p>Dimension : 40,7 x 42,9 x 19,7 mm</p>	<ul style="list-style-type: none"> <li>• Capacité à manipuler des charges lourdes.</li> <li>• Adapté à une variété d'applications</li> <li>• Fonctionne avec différents contrôleurs et microcontrôleurs comme Arduino.</li> <li>• Prix attractif</li> <li>• <b>Analogique et rétroaction en position complexe</b></li> </ul>

Finalement, nous avons opté pour le servomoteur Dynamixel AX-12A qui remplit tous nos critères de sélection. De plus, c'est un servomoteur très utilisé sur les projets de robots chiens. Pour l'alimentation des servomoteurs, nous allons utiliser un générateur basse fréquence.



Servomoteur AX-12A

Nous avons aussi acheté des fils connecteurs, du PLA pour l'impression, un adaptateur U2D2 pour pouvoir connecter dans un premier temps les servomoteurs au PC, puis ensuite pour connecter les servomoteurs à la Raspberry Pi 4, un accéléromètre MPU-6050 et aussi un petit module, le 74LS541 pour faciliter la connexion entre la Raspberry et les servomoteurs.



Connecteur U2D2 avec câble USB et câbles TTL



Accéléromètre MPU-6050

Ci-dessous, le budget prévisionnel établi afin de vérifier le respect du cahier des charges, les prix sont indiquées TTC et avec les frais de livraison en france métropolitaine inclus :

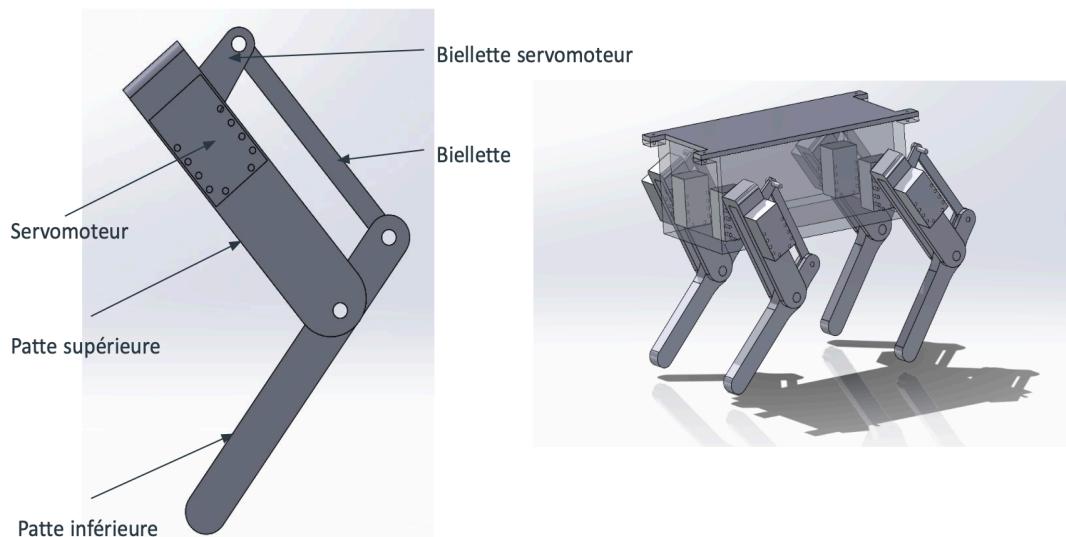
<b>Budget provisionnel :</b>			
Composants	Quantité	Prix TTC (€) *	
Fils connectique (Males-Femeles)	2	4,94	
Fils connectique (Males-Males)	2	4,94	
Raspberry Pi 4 Modèle B 4 Go	1	73,4	
Alimentation Raspberry Pi modèle B 4 go	1	17,09	
Servomoteur AX-12A (numériques + capteur position intégré)	10	596,52	
Connecteur U2D2 Dynamixel/PC	1	58,92	
Accéléromètre (Avec gyroscope)	2	47,75	
PLA Impression 3D (2,3 kg)	3	81,03	
SN74LS541 (communication carte-servos)	3	12,64	
Total		897,23	
Priorité faible			
Priorité moyenne			
Priorité élevée			

Budget prévisionnel

## Conception du robot

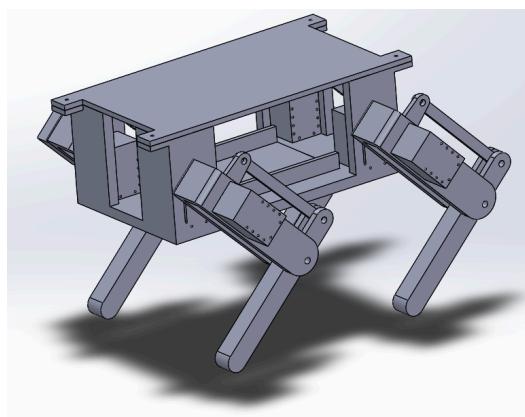
### Modélisation 3D

Une étape cruciale a été la réalisation d'une modélisation 3D du robot. Nous avons choisi d'utiliser une bielle pour pouvoir contrôler la patte inférieure du robot. Cette conception a été réalisée en utilisant le logiciel SolidWorks 2023, un outil puissant de modélisation 3D largement utilisé dans l'industrie pour ses capacités avancées et sa précision. Grâce à SolidWorks, nous avons pu créer une représentation virtuelle complète du robot, en tenant compte de chaque composant et de chaque détail, de sa structure globale à ses mécanismes internes. Ci-dessous, nous avons une première version de notre modèle 3D :



Modèle 3D de la patte et du robot v1

Cependant, nous avons ensuite modifié le corps du robot. En effet, nous avons enlevé de la matière afin d'alléger sa masse, de pouvoir intégrer et interchanger les composants plus facilement notamment grâce au socle positionné au centre du corps, et de faire entrer les câbles. Le nouveau modèle de notre robot est le suivant :

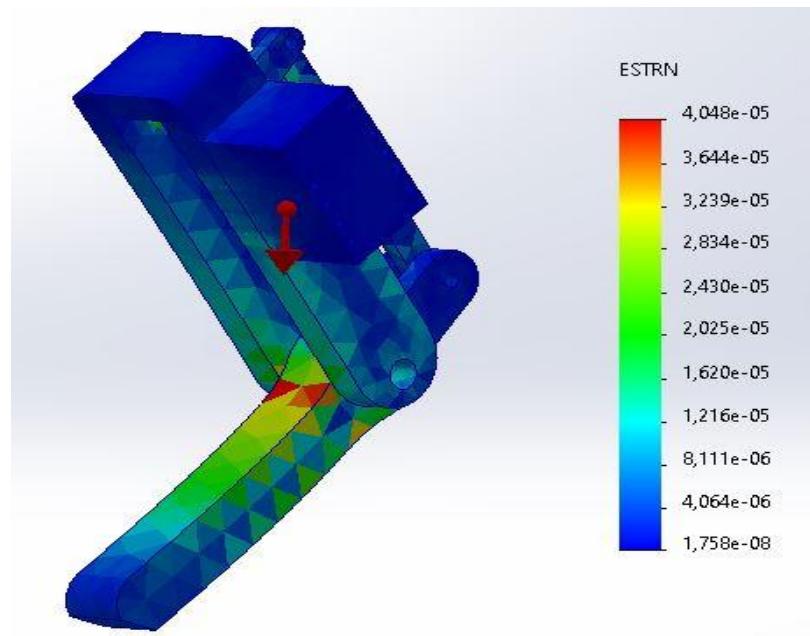


Modèle 3D du robot v2

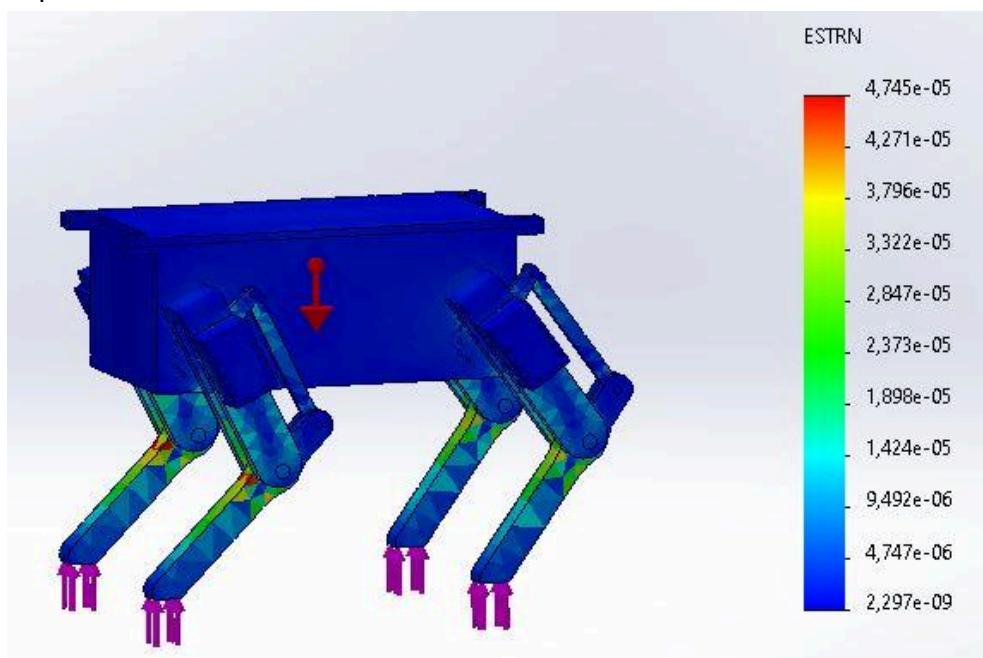
Nous passons maintenant à une étude mécanique en statique afin de valider notre modèle, et de passer à l'impression des pièces.

## Étude de déformation

Patte seule :

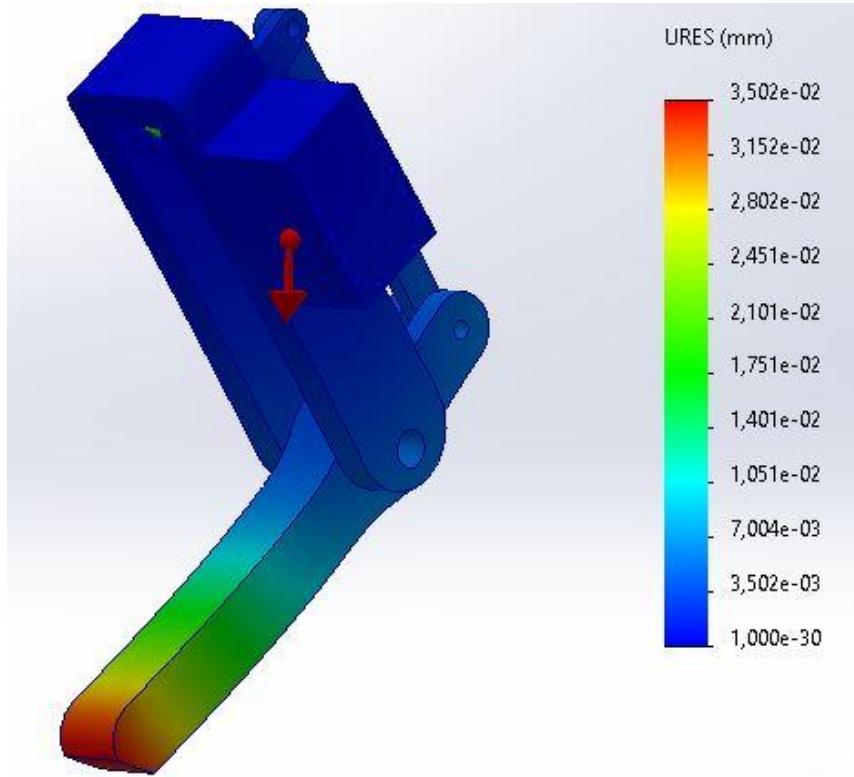


Robot complet :

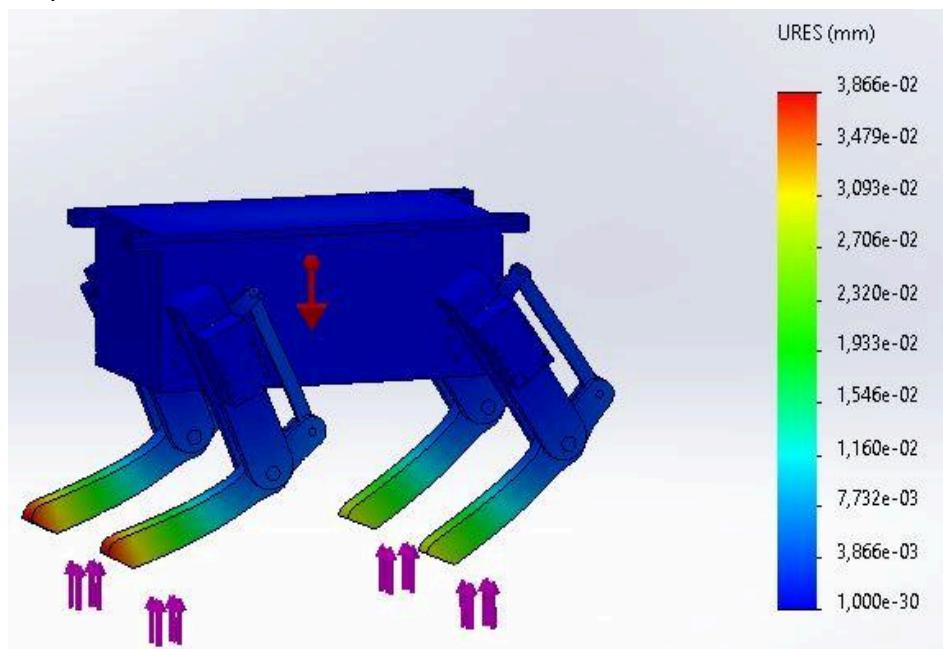


## Étude de déplacements

Patte seule :



Robot complet :

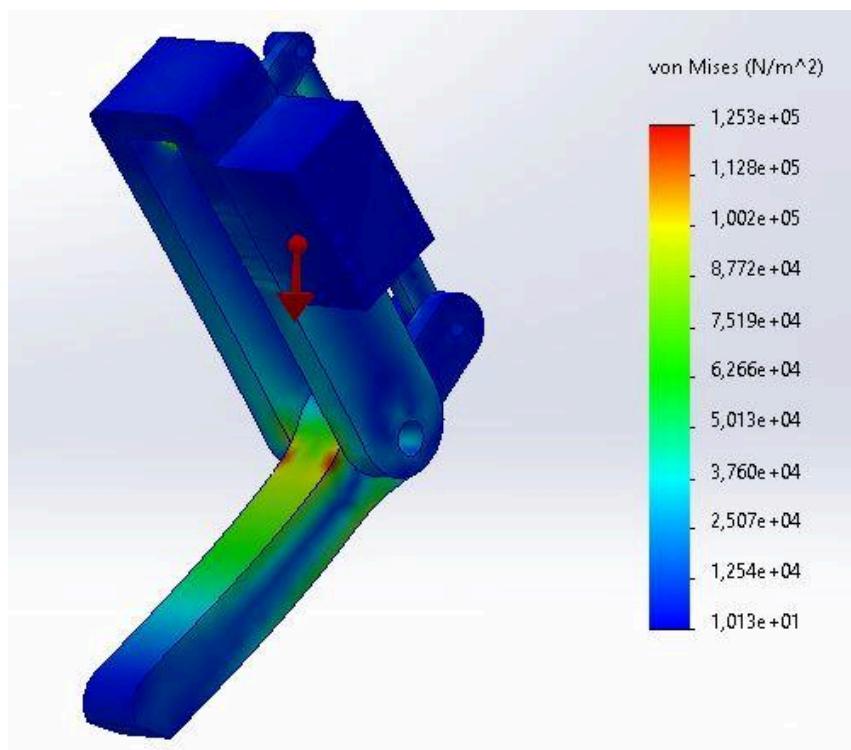


## Étude de contraintes

Nous avons également réalisé une analyse détaillée des contraintes auxquelles il pourrait être soumis. Cette analyse des contraintes vise à évaluer la résistance du robot et à identifier les zones de stress potentiellement critiques dans sa structure.

En utilisant les outils avancés de simulation de SolidWorks, nous avons modélisé différentes situations de charge qui pourraient affecter le robot pendant son fonctionnement.

Les résultats de notre analyse des contraintes ont permis d'identifier les zones de stress maximum dans la structure du robot, en mettant en évidence les zones potentiellement sujettes à des déformations ou à des ruptures sous l'effet des charges appliquées. Dans notre cas, nous avons soumis une patte au poids total du robot, ce qu'on peut supposer être le cas le plus défavorable. Ainsi, nous avons constaté que c'est l'intérieur de la patte comme montré dans la figure ci dessous (zone en rouge) qui est le plus sollicité. Le matériau que nous avons utilisé sur Solidworks à un module d'élasticité de 3 GPa contrairement à notre matériau qui a 2 GPa. Ainsi, nous sommes en mesure d'affirmer que si la simulation est validée avec ce matériau, alors il en est de même pour celui que l'on souhaite utiliser. En interprétant la contrainte maximale de von Mises c'est-à-dire en la convertissant en Gpa et en la comparant à notre limite de déformation, nous avons validé le matériau utilisé. En effet, nous pouvons maintenant garantir la sécurité et la durabilité du robot dans des conditions d'utilisation réalistes.



## Impression 3D

Après la modélisation 3D sur SolidWorks, nous avons lancé l'impression 3D de notre modèle auprès du service d'impression du LAMIH et du FABLAB. Le matériau utilisé pour l'impression est le PLA.

Le socle, le capot et le corps du robot ont été imprimés par le FABLAB (Mr Lesbros N.), et les pièces composant les pattes ont elles été imprimées par le LAMIH (Mr Lippert M.).



Ces premières impressions nous auront permis de retravailler le modèle 3D des pattes afin de limiter les frottements avec le corps du robot mais aussi d'adapter le diamètre des liaisons pivot au composants que nous utiliserons plus tard. De plus, le montage d'une patte temporaire nous aura permis de simuler le mouvement de celle-ci et donc de travailler sur le programme afin de rendre ce mouvement le plus fluide possible.

## Configuration de la Raspberry Pi 4

### Installation de l'OS et configuration

Dans un premier temps, il a fallu flasher la carte et, par le biais d'une **carte SD**, la paramétriser en **installant l'OS** pour qu'elle soit utilisable. Pour cela, nous avons suivi le PDF disponible sur le drive dans "7-Programmation".

#### Très important :

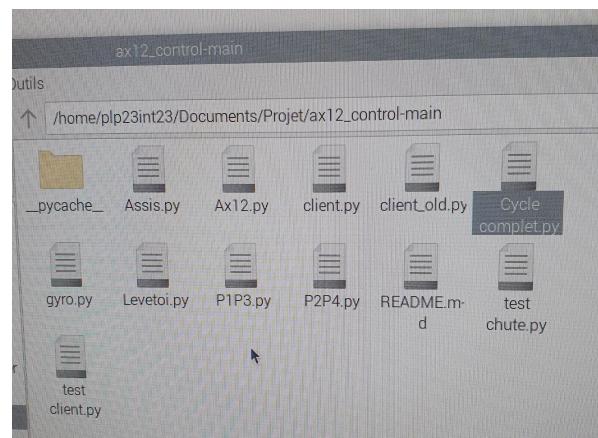
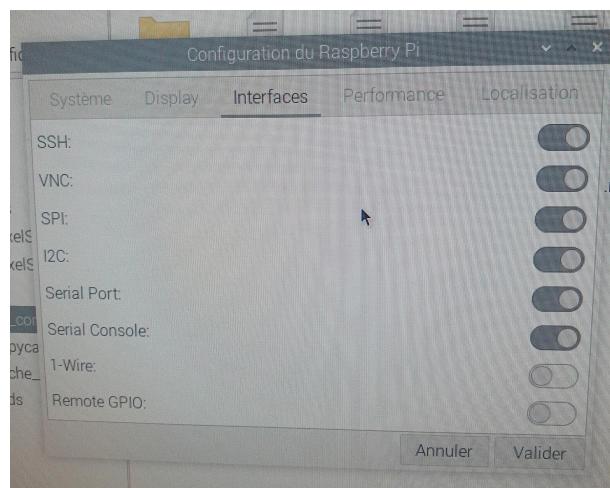
- **le nom de la Raspberry est : plp23int23**
- **le mot de passe est : plp23int23**

Pour ensuite travailler sur la Raspberry, plusieurs choix s'offre à vous :

- depuis l'invité de commande, (Terminal sur Mac OS) via le port **SSH** (à activer dans les réglages)
- brancher un **câble Micro-USB HDMI** pour travailler directement sur l'interface graphique
- À l'aide de **VNC Viewer**, en récupérant l'adresse IP de la Raspberry, cela affiche l'interface graphique de la Raspberry dans un onglet de votre ordinateur. (à activer dans les réglages).
- 

Le plus pratique est le **câble Micro-USB HDMI**, malheureusement, nous n'avions pas prévu cette dépense lors du choix des composants et nous nous sommes arrangés avec des professeurs pour en emprunter un pour la durée de notre projet.

Nous avons également activer le port I2C de la Raspberry (Logo Raspberry en haut à gauche -> Préférences -> Configuration du Raspberry Pi -> Interface) permettant d'utiliser notre MPU6050 (Accéléromètre, Gyroscope)



## Mise en place de la communication wifi

Un des critères du cahier des charges est qu'il sera **commandable par Wi-Fi**, nous devons donc mettre en place cette communication sans fil avec l'emploi d'un script serveur/client.

Il faut d'abord s'assurer que les deux appareils sont sur le même réseau WiFi, sinon la communication ne s'effectuera pas. Ensuite, en utilisant Python avec la bibliothèque 'socket'. Cette bibliothèque permet de gérer la communication réseau.

Ensuite, dans le code Python, on divise les tâches en deux parties : le serveur (ici notre PC) et le client (ici notre Raspberry).

Une fois que tout est en place, on peut commencer à échanger des données. On peut imaginer ces données comme des messages qu'on se passe entre l'ordinateur et la Raspberry Pi. Le serveur (l'ordinateur) reçoit le message du client (la Raspberry Pi) et peut y répondre.

Bien sûr, il faut penser à gérer les erreurs, comme par exemple si la connexion est perdue ou si le message est mal écrit. Ça permet de rendre notre programme fiable et robuste.

En suivant ces étapes et en écrivant le code nécessaire, on peut créer une communication WiFi entre l'ordinateur et la Raspberry Pi, leur permettant d'échanger des données facilement et de manière fiable.

### Codes : se referer aux codes “serveur.py” et “client.py”

## Création du serveur virtuel et connexion avec la Raspberry

Le serveur virtuel sera le PC réalisant tous les calculs tandis que le client sera la Raspberry recevant les consignes à envoyer aux servomoteurs. Pour ce faire, la Raspberry ainsi que l'ordinateur faisant office de serveur doivent être connectés aux même réseau wifi, la liaison se fait ensuite via l'adresse IP du réseau auquel ils sont connectés. Ce lien conduit à une vidéo montrant cette étape de connexion Wifi : <https://youtu.be/546fmn2v-rM>

Prenez en compte que dans les codes “**serveur.py**” et “**client.py**”, il faut absolument changer l'adresse IP du serveur et mettre celle de votre ordinateur sinon cela ne marchera pas.

Ensuite nous lançons le serveur, il est important de le lancer depuis un Terminal comme dans la vidéo et non dans un environnement IDLE Python à cause de l'asynchronisation.

Une fois le serveur lancé via le script “**serveur.py**” avec des commandes bashs, il faut lancer ensuite le “**client.py**”. Un message de confirmation de connexion est présent et sera affiché sur le Terminal du serveur.

Par la suite le serveur peut envoyer 3 instructions que l'on a programmé dans le script "client.py" :

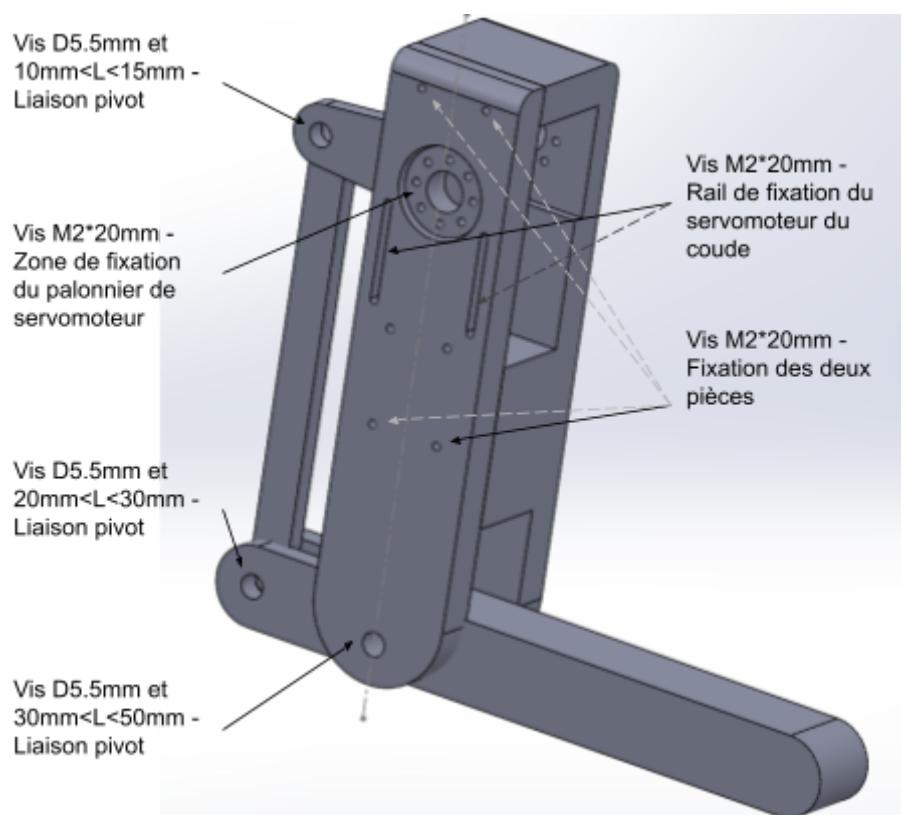
- debout : lève le robot
- assis : assis le robot
- marche : lève le robot, effectue 3 cycles de marche, s'assoit

**Attention, nous n'avons pas eu le temps de mettre des positions maximales pour les servomoteurs causées par notre conception du robot. Vous serez amené à créer d'autres instructions et de ce fait, à dépasser certains angles, des fils peuvent donc se sectionner et/ou des servomoteurs peuvent cesser de fonctionner dû à un trop fort couple résistant.**

## Début du prototypage du robot

### Prototypage du robot

Nous pouvons maintenant commencer le prototypage. Pour ce faire nous assemblons les divers éléments de la patte en impression 3D, à commencer par les pattes afin de réaliser des tests de coordination du mouvement.

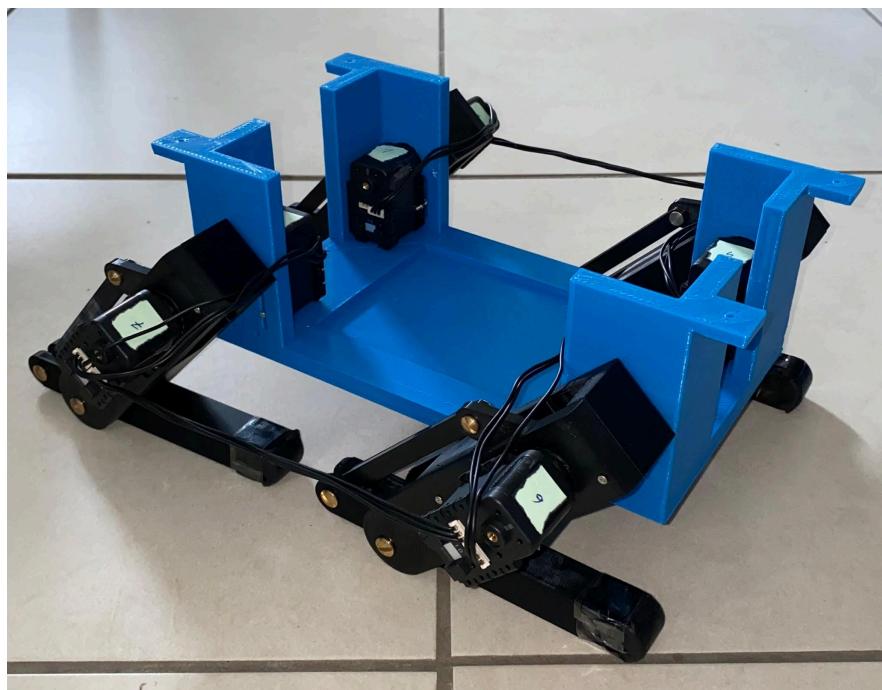


Comme stipulé sur la figure ci-dessous nous utilisons des vis M2\*20mm pour le montage des pièces. Les liaisons pivot entre les composant est réalisé grâce à des vis relieur en laiton :



Attention lors des phases de simulation sur le prototype réel, **les vis peuvent prendre du jeu et se dévisser au fur et à mesure de l'utilisation**. Il faudra donc penser à utiliser de la loctite ou autre pour bloquer les vis en rotation ou bien simplement penser à les visser assez souvent s'il est prévu de démonter ces axes dans le futur.

Une fois les pattes montées, nous montons les servomoteurs à l'intérieur de la base en utilisant le même principe que précédemment avec des rails de fixation. Une fois cette étape passée nous montons chaque patte sur le robot et obtenons alors le prototype du robot chien, sans électronique embarquée :



Une fois cette étape réalisée nous commençons déjà la simulation du programme sur le robot, mais dans cette partie nous terminerons par le prototypage complet du robot avec tous les composants nécessaires.

Décrivons maintenant le câblage des composants à mettre en place sur le prototype. Tout d'abord, l'alimentation de la Raspberry Pi se fait grâce au câble d'alimentation 5V 3A. Ensuite, l'adaptateur U2D2 se branche par USB à la Raspberry et est connecté au servomoteur d'identifiant 1 via un câble TTL 3 voies.

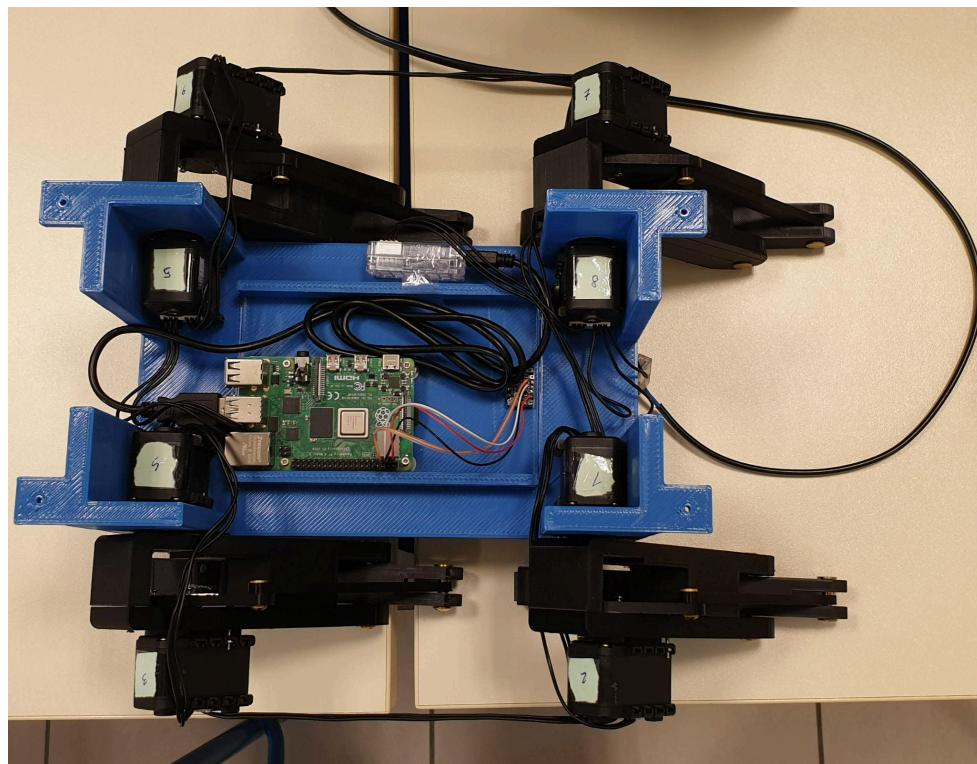
Les 8 servomoteurs sont branchés entre eux en série (daisy-chain) dans l'ordre croissant des identifiants (ID 1 à 8).

Enfin, le dernier servomoteur d'identifiant 8 est connecté au générateur avec une tension d'alimentation de 11,1 V (comme recommandé sur le site Génération Robots) comme ci-dessous :

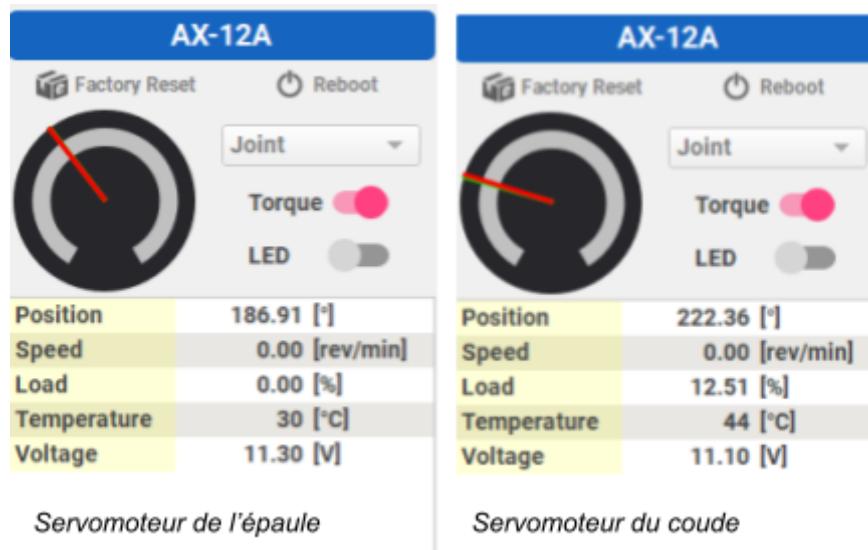


Ce générateur fournissant maximum 30V/3A par source (deux sorties CH1 et CH2) nous utilisons sa fonctionnalité intégrée de mise en parallèle des sources. Cela nous permet d'obtenir une source délivrant 30V/6A, le courant absorbée par un servomoteur sera alors de maximum 750mA. Un servomoteur ayant besoin de 600mA à 900mA en fonctionnement nominal, utiliser une seule source pour 8 servomoteurs n'était pas possible.

Une vue du dessus du robot permet de mieux éclaircir la description du câblage :



Le U2D2 permet de communiquer en full-duplex avec la raspberry afin de récupérer les données du servomoteur en temps réel. Il est par exemple possible de récupérer la charge sur les servomoteurs, leurs positions ou encore leurs vélocité :



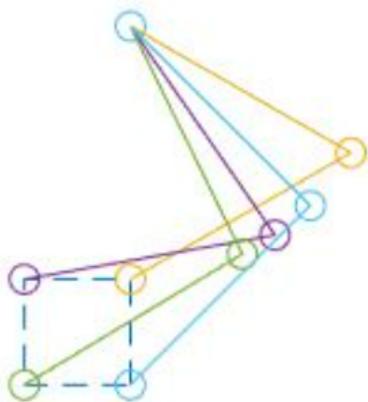
Il est très important d'avoir un retour d'informations de tout cela en cas de débogage ou d'optimisation. Connaître la charge des servomoteurs et leurs positions est un vrai luxe que nous permette ces servomoteurs pour savoir si le robot atteint les bonnes positions, leurs précisions et si le robot est trop lourd. Cependant, il est également possible d'utiliser le 74LS541 pour la même communication mais il faudrait le brancher au pin associé sur la raspberry. Dans notre situation, par manque de temps et par facilité, nous avons décidé d'utiliser le U2D2 à la place car il suffisait simplement de le brancher en USB à la raspberry. L'objectif ici est que, par le biais de bielles, la rotation du servomoteur du coude puisse engendrer le mouvement de la patte inférieure. Pour cela, les liaisons pivots permettent de fluidifier le mouvement et de transmettre au mieux la rotation du servomoteur. Il est plutôt envisageable d'utiliser des roulements par la suite. Il a donc fallu ajuster les diamètres des trous prévus à cet effet.

Une fois que la communication entre les servomoteurs est optimale et que la daisy chain est prête à l'emploi, nous pouvons tester le fonctionnement d'une patte en rédigeant le code de marche.

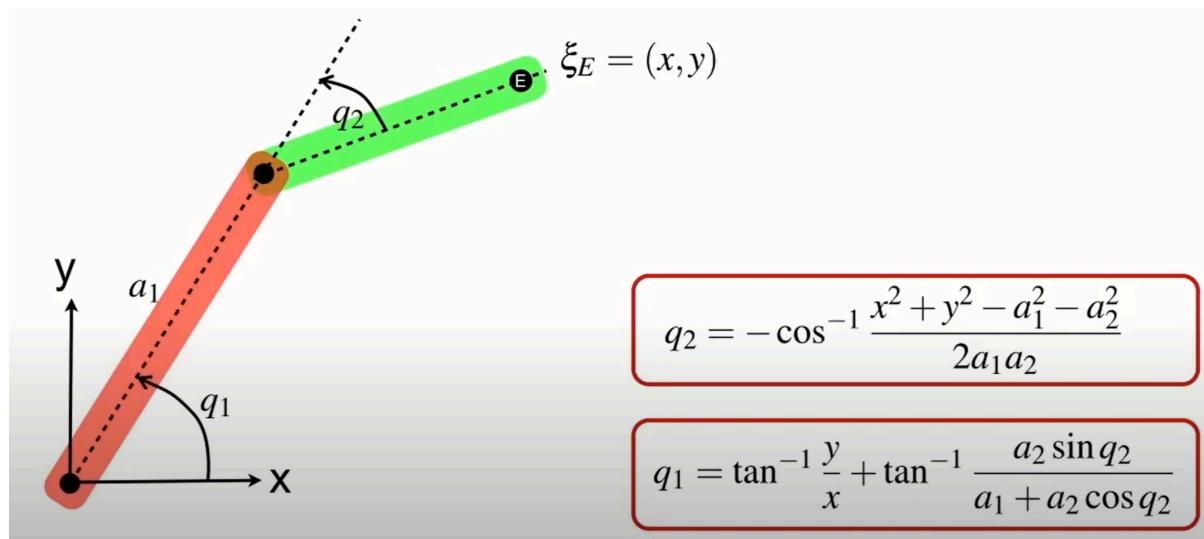
## Principe du cycle de marche

Pour pouvoir effectuer un cycle de marche de 4 pas, le mouvement de pattes de notre robot doit se baser sur le principe de la cinématique inverse qui désigne les méthodes de calcul des positions et rotations d'un modèle articulaire afin d'obtenir une position désirée.

Dans notre cas, nous souhaitons que chaque patte du robot suive une trajectoire sous la forme d'un carré comme ci-dessous :



Les équations d'angles, entre la base du robot et la partie haute de la patte  $q_1$ , et entre la partie haute et basse de la patte  $q_2$ , qui régissent le mouvement des pattes sont les suivantes :



[5]

Dans notre cas, nous avons le bras vers le bas c'est pourquoi nous avons fait une rotation du repère de 90 dans le sens horaire. Ainsi, x vertical vers le bas et y horizontal vers la droite. Suite à cela, il fallait déterminer les coordonnées des 4 points du carré (voir code python cinématique inverse) que le robot devait atteindre afin d'utiliser les 2 formules pour obtenir leurs angles associés.  $q_1$  correspondant au servomoteur de l'épaule et  $q_2$  celui du coude.

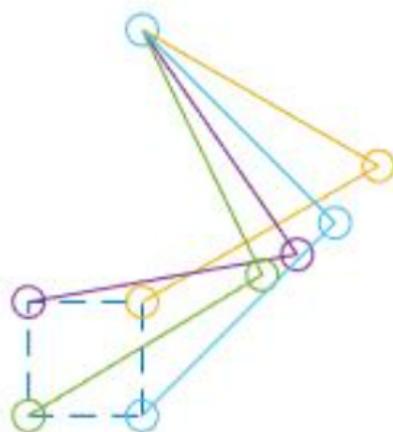
Afin de vérifier les équations utilisées, nous avons écrit un script Matlab pour simuler la position de la patte du robot sur chaque coin du carré :

```
% Simulation pour chaque coin du carré
for i = 1:size(square_corners,1)
    % Coordonnées du coin actuel
    x1 = square_corners(i, 1);
    y1 = square_corners(i, 2);

    % Calcul des coordonnées du point P2 (coude)
    q2 = acosd((x1^2 + y1^2 - a1^2 - a2^2) / (2 * a1 * a2));
    q1 = atan2d(x1, -y1) - atan2d((-a2 * sind(q2)), (a1 + a2 * cosd(q2)));

    % Coordonnées du coude (point P2)
    x2 = x1 - a1 * sind(q1);
    y2 = y1 + a1 * cosd(q1);
```

La simulation nous donne la trajectoire voulue pour le mouvement de patte du robot :



Après avoir vérifié les formules utilisées, il nous faut maintenant trouver les angles  $q_1$  et  $q_2$  pour notre application. Pour cela, nous avons entré les formules de  $q_1$  et  $q_2$  sous Python :

```
def formule(x, y):
    q2 = np.arccos((x**2 + y**2 - a1**2 - a2**2) / (2 * a1 * a2))
    q1 = np.degrees(np.arctan2(-y, x)) - np.degrees(np.arctan2((a2 * np.sin(q2)), (a1 + a2 * np.cos(q2))))
    return q1, np.degrees(q2)
```

Les angles en degrés  $[q_1, q_2]$  de la patte calculés pour chaque coin du carré sont les suivants :

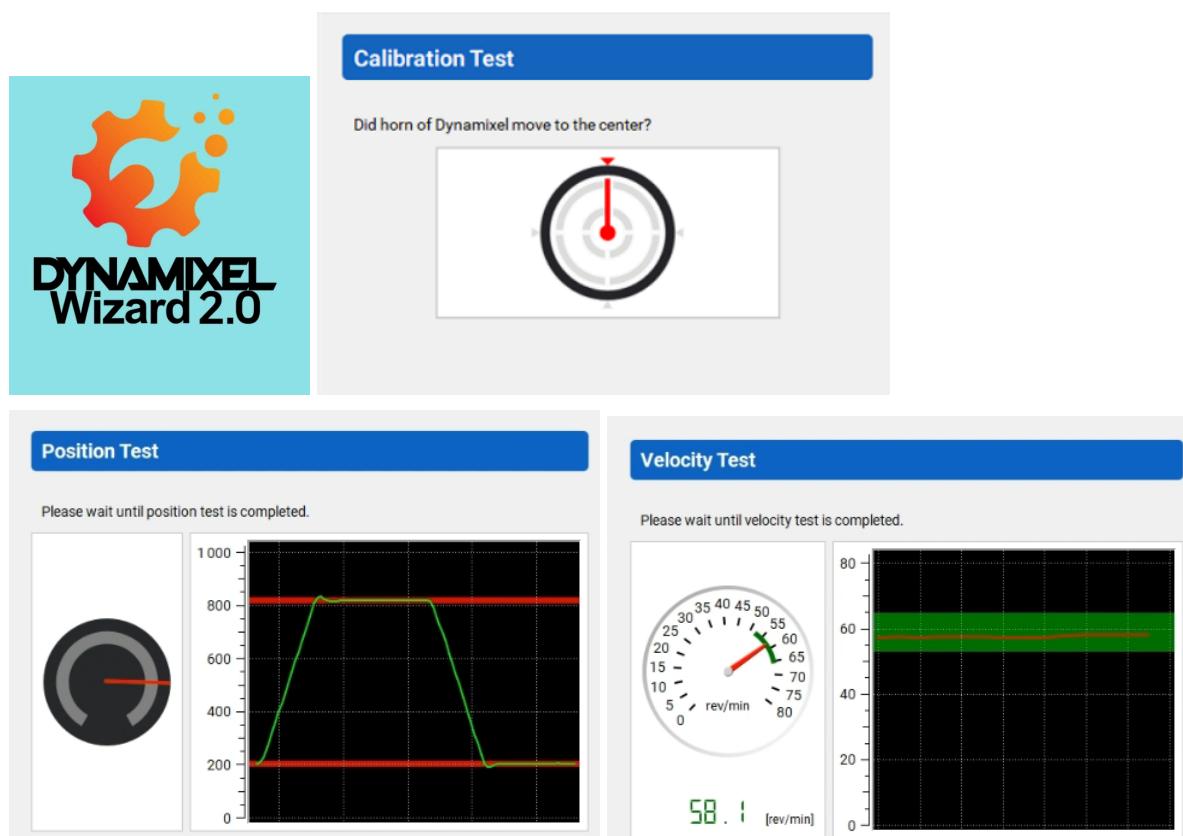
```
'formule pour x1, y1 : (-44.99999999999999, 89.99999999999999)
'formule pour x2, y2 : (-60.08111491357978, 120.16222982715956)
'formule pour x3, y3 : (-34.61008663325266, 114.55991278712236)
'formule pour x4, y4 : (-26.093629916928926, 85.02014058102877)
'formule pour x1, y1 : (-44.99999999999999, 89.99999999999999)
```

## Programmation

### Vérification et configuration des servomoteurs

Connaissant les angles  $q_1$  et  $q_2$  des pattes du robot, il faut désormais mettre en marche les 8 servomoteurs AX-12A.

Pour cela, nous devons d'abord réaliser un “self-diagnosis” de chaque servomoteur afin de valider leur conformité, ce “self-diagnosis” comporte un test de calibration, un test de position et un test de vitesse. Il faut aussi attribuer un numéro d'identifiant unique à chaque servomoteur de 1 à 8. Tout cela se fait grâce au logiciel DYNAMIXEL Wizard 2.0 .[\[6\]](#)



Une fois la configuration des servomoteurs effectuée, nous leur attribuons chacun un ID différent (dans notre cas de 1 à 8) afin de pouvoir communiquer avec le servomoteur désiré en les appelant par leur ID dans le futur programme. Le langage de programmation utilisé sera Python, cette programmation se base d'abord sur un tutoriel de prise en main des servomoteurs.[\[7\]](#) Pour cela, nous avons importé la librairie de Dynamixel : DYNAMIXEL SDK. [\[8\]](#) Cette librairie fournit des fonctionnalités de contrôle du AX-12A. Voici un lien conduisant à une vidéo montrant l'étape complète du contrôle de conformité d'un servomoteur : <https://youtu.be/LGhUQVvqKMI>

## Rédaction du cycle de marche

**Codes :** voir le chemin d'accès de la Raspberry :

“/home/plp23int23/Documents/Projet/ax12\_control-main” + drive

“7-Programmation/Pycharm : code qui marche Ax12”

Pensez à lire les README sur la raspberry.

### **Indications :**

P1 : patte arrière gauche

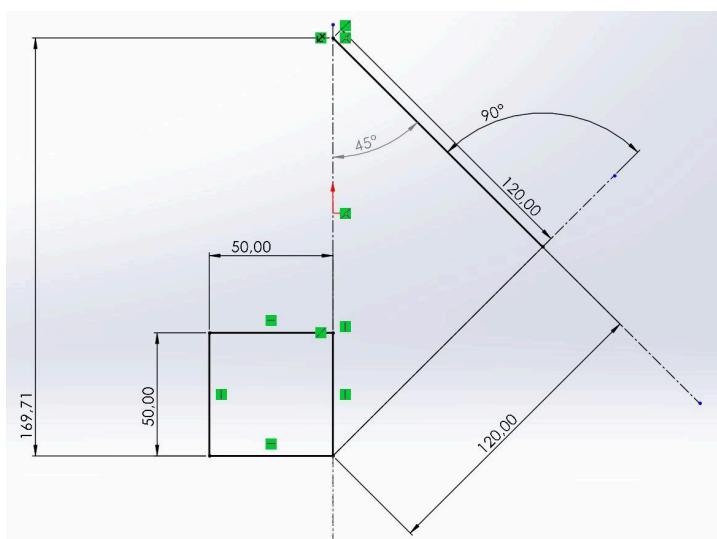
P2 : patte avant gauche

P3 : patte avant droite

P4 : patte arrière droite

Une seule patte

Pour commencer, nous utilisons la méthode vue précédemment afin de définir les angles à atteindre pour un mouvement carré, nous les incorporons ensuite dans le futur code de marche :



Une fois cela fait, il faut faire en sorte de synchroniser le servomoteur de l'épaule avec celui du coude. Pour ce faire, nous avons simplement défini le servomoteur de l'épaule comme 2 fois moins rapide que celui du coude (le servomoteur du coude réalisant des rotations moins amples que celui de l'épaule).

Nous avons alors testé un cycle de marche pour une seule patte, ce lien conduit à une vidéo du résultat obtenue : <https://youtube.com/shorts/zE527YKfpEU>

## Deux pattes

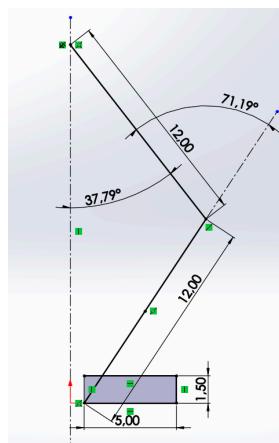
Le résultat étant concluant, nous avons décidé de tester sur 2 pattes. C'est à partir de ce moment qu'il faut être attentif. En effet, pour notre cas, nous avons opté pour une marche faisant bouger 2 pattes en diagonales. C'est-à-dire que la patte arrière gauche bouge simultanément avec la patte avant droite (pattes opposées) puis c'est aux tours des 2 autres pattes en simultanée. En prenant cela en compte, nous constatons un mouvement miroir. En d'autres termes, vous ne pouvez pas mettre les mêmes angles pour les pattes gauches et les pattes droites car elles sont symétriques. C'est pourquoi il a fallu faire de nouvelles conversions d'angles pour les pattes de droite. Une fois ces conversions faites, il est possible de tester 2 pattes en même temps.

Lors des instructions de goal position pour les servomoteurs, nous nous sommes rendu compte qu'il ne fallait pas simplement les écrire dans l'ordre des servomoteurs sur le bus (daisy chain). Lorsque l'on fait cela, il y a le dernier servomoteur qui est en retard. C'est pourquoi nous avons défini un ordre de priorité que ce soit pour 2 pattes ou 4 pattes. On donne cette instruction d'abord sur les servomoteurs de chaque épaule (1, 4, 5 et 8) puis ceux sur les coudes (2, 3, 6 et 7). En faisant cela, ils s'exécutent tous exactement en même temps. Voici une démonstration vidéo illustrant nos propos : <https://youtube.com/shorts/HCBnhl800ys?feature=share>. Une fois ce problème résolu, le code final pour 2 pattes a été rédigé (voir les codes python). L'un pour les pattes P1P3 et l'autre pour les pattes P2P4.

## Quatres pattes

Puisque 2 pattes fonctionnent, on se base exactement sur le même principe et en utilisant les mêmes angles associés pour les pattes droites et gauches (on réutilisent les codes P1P3 et P2P4 que l'on combine).

Etant donné que nous avions 2 pattes opposées en mouvement puis les 2 autres, cela faisait que le robot penchait trop vers l'arrière car il tenait en équilibre sur 2 pattes lors de la transition. Nous avons alors décidé de lancer les prochaines pattes avant que celles en mouvement ne terminent leur cycle. Ainsi le robot est devenu légèrement plus stable. Mais il semblait tout de même y avoir un problème au niveau de la répartition du poids. C'est pourquoi, nous avons opté pour quelques changements. En effet, la trajectoire carré de la patte rendait le robot instable et provoquait sa chute. Nous avons alors décidé de modifier cette trajectoire par un rectangle, et aussi de remonter la patte à l'état initial, optant pour une meilleure stabilité :



Nous calculons alors les nouveaux angles de consignes, toujours grâce à l'emploi de la méthode décrite [précédemment](#). Nous obtenons alors des valeurs d'angles que nous convertissons afin de les utiliser dans notre programme. Cette conversion est nécessaire car la rotation du servomoteur s'effectue entre  $0^\circ$  et  $300^\circ$  mais les consignes données sont comprises entre 0 et 1023 au sein du programme. La valeur de référence du servomoteur est en son milieu qui est  $150^\circ$  soit 512 après conversion. Nous obtenons alors les consignes suivantes pour le nouveau cycle de marche :

```
angle_m1_conv = [641, 678, 702, 661, 641]
angle_m2_conv = [755, 731, 772, 794, 755]
angle_m1_mirr_conv = [383, 346, 322, 363, 383]
angle_m2_mirr_conv = [269, 293, 252, 230, 269]
```

On définit :

- “m1” pour le servomoteur de l'épaule et “m2” pour celui du coude
- “conv” pour conversion et “mirr\_conv” pour conversion sur les pattes miroir donc droites.

Lors des premiers tests, nous observons une nette amélioration de la stabilité, le robot ne chute plus et est désormais capable de se déplacer.

Nous avons ensuite implémenté une boucle while afin de réaliser plusieurs cycles de marche à la suite mais nous avons observé des saccades sur le prototype à la fin de chaque cycle. En effet à chaque nouvel itération de la boucle tous les servomoteurs s'arrêtaient avant de reprendre leur cycle, ce temps d'arrêt venait entacher le cycle de marche du robot. Pour notre prototype nous avons donc décidé de rédiger à la main 3 cycle de marches, ce qui alourdit le programme mais rend notre cycle de marche beaucoup plus fluide. De manière générale la marche de robot ne s'effectue pas par la rédaction d'un programme conventionnel en donnant des consignes à chaque servomoteurs, le résultat n'est donc pas à l'image d'autre robot présent sur le marché, mais c'est néanmoins le point qui sera travaillé à l'avenir.

### En cas de chute

Dans cette partie nous utilisons le MPU6050 afin de permettre au robot de réagir en cas de chute et donc de limiter la casse. Pour ce faire, nous définissons une valeur critique synonyme de chute du robot. Cette valeur est constamment comparée aux valeurs mesurées en temps réel et lorsqu'elle est atteinte le robot rétracte ces pattes en position assise. Cela lui permettra par la suite de relancer son cycle de marche étant donné qu'il sera de nouveau en position initiale. Voici une vidéo qui représente ce cas de figure : [https://youtu.be/Ub\\_7GP6twuI](https://youtu.be/Ub_7GP6twuI)

## Proposition d'un programme basée sur l'IA

Voici le matériel mis à notre disposition :

- **Servomoteurs AX12 de Dynamixel** : Utilisés pour le mouvement du robot, permettant une gamme de mouvements précise et contrôlable.
- **MPU6050** : Un capteur combinant un accéléromètre et un gyroscope, utilisé pour surveiller l'orientation et le mouvement du robot.
- **Raspberry Pi** : Sert de plateforme de contrôle, exécutant l'algorithme d'apprentissage par renforcement et communiquant avec les servomoteurs et le MPU6050.
- **Python et bibliothèques associées** : Le code est écrit en Python, avec l'utilisation de bibliothèques telles que gym pour définir l'environnement d'apprentissage par renforcement, stable\_baselines3 pour l'implémentation de l'algorithme SAC (Soft Actor-Critic), et des bibliothèques spécifiques pour interagir avec le matériel (AX12, mpu6050).

Le Soft Actor-Critic (SAC) est choisi pour sa capacité à gérer des espaces d'actions continus et sa promotion de l'exploration efficace grâce à l'entropie. L'algorithme ajuste les actions du robot pour maximiser la récompense, basée sur la distance parcourue et la stabilité du robot.

Nous avons implémenté 3 fonctions de base :

**apply\_action** : Applique les actions décidées par l'agent SAC aux servomoteurs, en transformant les valeurs d'action dans l'espace [-1, 1] en positions valides pour les servomoteurs.

**get\_observation** : Récupère et renvoie l'état actuel du robot, comprenant les positions des servomoteurs et les données du MPU6050.

```
def get_observation(self):
    # Récupération des positions des servomoteurs
    servomotor_positions = [motors[motor_id].get_present_position() for motor_id in motor_ids]

    # Lecture des données de l'accéléromètre et du gyroscope
    accelerometer_data = mpu.get_accel_data()
    gyro_data = mpu.get_gyro_data()

    # Combinaison des données dans un seul vecteur d'observation
    observation = np.array(servomotor_positions +
                           [accelerometer_data['x'], accelerometer_data['y'], accelerometer_data['z'],
                            gyro_data['x'], gyro_data['y'], gyro_data['z']])
```

**calculate\_reward** : Calcule la récompense en fonction de la vitesse angulaire (pénalité pour chute) et de la distance parcourue, avec une récompense supplémentaire pour la stabilité.

```
# Initialisation de la récompense
reward = 0
done = False

# Pénalité pour avoir dépassé le seuil de vitesse angulaire (chute)
if gyro_speed > 15:
    reward -= 100 # Malus important indiquant une chute
    done = True

# Récompense pour la distance parcourue
# Plus le robot se déplace rapidement dans la bonne direction, plus il gagne de points
reward += distance

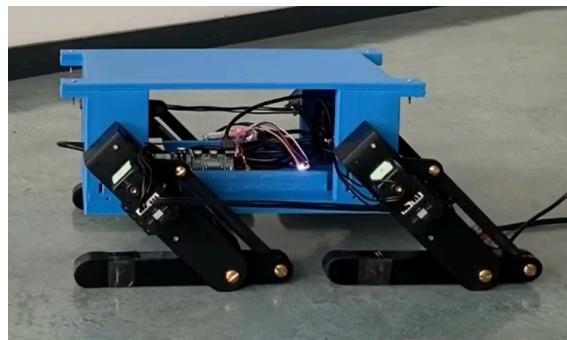
# Récompense pour stabilité
# Encourage le robot à maintenir son équilibre avec une vitesse angulaire faible
stability_reward = 1 - (gyro_speed / 15) # Plus la vitesse angulaire est proche de 0, plus la récompense est élevée
reward += max(stability_reward, 0) # Assure que la récompense de stabilité n'est pas négative
```

Le système de récompense est conçu pour encourager le robot à marcher le plus loin possible sans tomber, en mettant l'accent sur la vitesse et la stabilité. Une pénalité est imposée si le gyroscope détecte une vitesse angulaire élevée, indiquant une chute potentielle. Des récompenses sont attribuées pour le déplacement rapide et la conservation d'une faible vitesse angulaire, favorisant ainsi une marche stable.

**Attention, il faut prendre en compte que cela n'est qu'une proposition d'un algorithme d'entraînement, qu'il comporte probablement des erreurs et qu'il doit être améliorer**

## Prototype final du robot

Pour finir nous peaufinons notre programme pour ajouter par exemple des fonctions supplémentaires permettant un mouvement plus réaliste. Par exemple, le robot commence par se lever aux positions initiales calculées préalablement, puis il réalise son cycle de marche et à la fin de celui-ci il retourne en position assise. Voici les diverses positions :



Remarque : avant de mettre le robot en position initiale ou de bouger les pattes manuellement, il faut **impérativement** que les servomoteurs **ne soient pas alimentés** ! autrement, vous risquez de forcer sur les servomoteurs et d'atteindre la charge maximum qu'ils peuvent supporter au risque de les abîmer. Il est donc fortement conseillé de mettre le robot en position initiale manuellement seulement si les servomoteurs ne sont plus alimentés ou d'utiliser le code “Assis.py”.

Voici donc un lien conduisant à une vidéo du robot réalisant un cycle de marche complet :  
<https://youtu.be/PMTcuTkGsEU>

Comme nous le constatons sur la vidéo, il reste un problème car la patte arrière droite ne fait pas le mouvement rectangle et frotte le sol. Cependant, lorsque le robot est relevé dans les airs et que nous exécutons le cycle de marche, la trajectoire de la patte est bien conforme à la consigne. Néanmoins, cette méthode de rédaction de cycle de marche pour un robot n'étant pas conventionnelle, c'est sûrement l'origine de ce problème.

# Conclusion

En reprenant la description du cahier des charges de notre plateau-projet, nous pouvons affirmer que nous avons respecté tous ses critères :

- Conception d'un robot quadrupède d'une hauteur moyenne de 20 cm ✓
- Robot capable de réaliser un cycle de marche en ligne droite ✓
- Capable de se relever ✓
- Le robot sera commandé par Wi-Fi ✓
- Proposition d'un programme basé sur l'apprentissage par renforcement ✓
- Le budget alloué au projet ne devra pas excéder 1000€ TTC ✓
- Avant le 23/02/24 ✓

Néanmoins, ce projet de robot quadrupède marcheur reste perfectible.

En effet, si ce projet était à refaire, la présence d'un étudiant en filière ME dans l'équipe aurait été primordiale pour une étude mécanique approfondie du robot, notamment pour la stabilité du robot. Également, la présence d'un étudiant en filière ICY aurait été importante pour la programmation du cycle de marche (optimisation du code) et l'intégration de l'apprentissage par renforcement (IA).

Les améliorations possibles de notre robot sont l'optimisation du modèle 3D de notre robot pour une meilleure robustesse, l'utilisation de connectiques plus longues (notamment les câbles TTL) entre les servomoteurs pour tester d'autres prototypages possibles (par exemple, mouvements plus amples des pattes), et l'utilisation d'une protection pour les servomoteurs extérieurs, notamment en cas de chute du robot pour éviter la casse des servomoteurs.

Pour les futures équipes qui travailleront sur ce PLP, l'intégration et entraînement de l'IA est indispensable pour apprendre au robot à marcher grâce au système de récompense, et il pourra être intéressant de songer à intégrer un LIDAR pour cartographier l'environnement du robot avec une grande précision.

## Bibliographie

[1] : Robot Cheetah 3 :

<https://www.futura-sciences.com/tech/actualites/robotique-cheetah-3-prouesses-robot-55289/>

[2] : Robot ANYmal :

<https://www.anybotics.com/robotics/anymal/>

[3] : Robot Laikago :

<https://www.leblogdomotique.fr/robotique/unitree-robot-quadrupede-9471>

[4] : Robot HyQReal :

<https://hyq-real.eu/>

[5] : Équation des angles de marche :

<https://www.youtube.com/watch?v=IKOGwoJ2HLk&t=309s>

[6] : Logiciel Dynamixel Wizard 2.0 :

[https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel\\_wizard2/](https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_wizard2/)

[7] : Tutoriel de prise en main des AX-12A sous Python :

[https://www.youtube.com/watch?v=SqKS\\_cFuPh0](https://www.youtube.com/watch?v=SqKS_cFuPh0)

[8] : Librairie DYNAMIXEL SDK :

[https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel\\_sdk/overview/](https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/overview/)