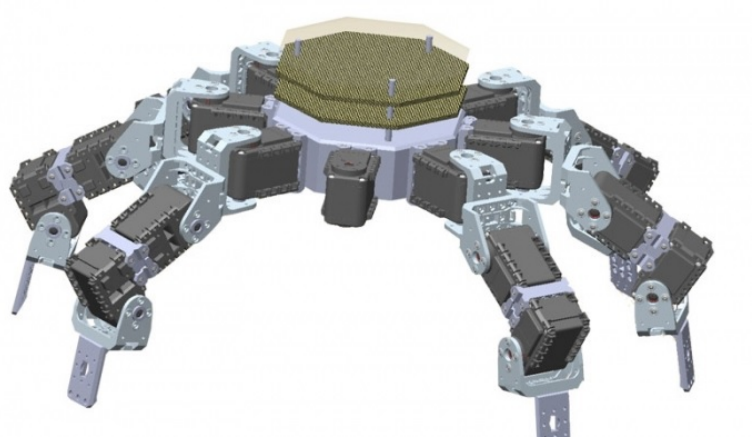
École Normale Supérieure Paris-Saclay
Université Paris-Saclay

Robot d'apprentissage MIMI

TER - Travaux encadrés de recherche



PECQUEUX-GUEZENEC Charly
ZUNIGA VILCAPOMA Deyvi

Département d'Électronique, Électrotechnique et Automatique.

Table des matières

Introduction

De nos jours, l'intelligence artificielle est devenue un outil très important pour stimuler le développement de disciplines, par exemple, la robotique qui utilise l'IA pour apprendre aux robots à s'adapter à certaines situations et, par conséquent, à aider les humains . Ces applications sont remarquables aujourd'hui parmi lesquelles nous avons des bras robotiques qui aident dans les usines de construction, des véhicules autonomes qui peuvent limiter le trafic et les accidents, des drones qui peuvent faire un service de livraison, des humanoïdes comme " Sophia " développé par Hanson Robotics et des robots pouvant se trouver dans des endroits dangereux pour l'être humain. Un exemple de ce dernier est le robot Atlas développé par Boston Dynamics pour effectuer des tâches de sauvetage.

L'objectif de ce TER est de donner aux participants une premier approche à l'intelligence artificielle en appliquant les principes de l'IA à un robot hexapode pour lui apprendre à marcher. Pour réaliser ce travail, nous allons commencer par modéliser le robot pour simuler l'IA et voir la faisabilité du projet et l'apporter au vrai robot.

1 Robot MIMI

Le robot MIMI est un outil d'enseignement qui a été construit comme solution exacte de la coupe de France de robotique 2011. Ce robot est un hexapode qui de compose de 18 servomoteurs AX12 numériques en réseau et reliant 6 pattes indépendantes.

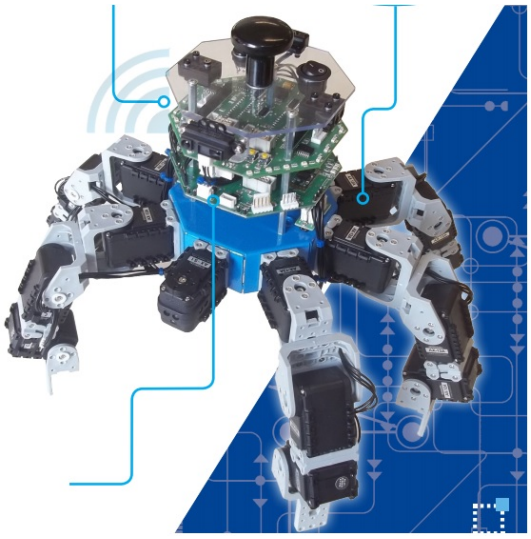


FIGURE 1 – Robot MIMI

Pour la réalisation de ce TER, nous avons pris la partie mécanique de ce robot et nous avons ajouté une carte de développement STM32 Nucleo-144 pour gérer la commande des moteurs et la Jetson Nano qui gère la partie apprentissage par renforcement.

1 Servomoteur AX12

Le servomoteur utilisé est le Dynamixel d'entrée de gamme AX-12A (figure 2) autorisant un fonctionnement en rotation continue ou de 0 à 300°. Ce moteur permet un retour d'information de la position, température, couple et tension d'alimentation. Ces données seront d'importance pour faire une analyse d'énergie pour bien choisir la batterie pour commander les 18 moteurs.

FIGURE 2 – Servomoteur AX12

Ces moteurs ont deux connecteurs 3 broches pour les connecter en cascade et ainsi faciliter la commande de ceux-ci. On utilise ce mécanisme en 3 moteurs pour chaque patte et la commande individuel sera effectuée grâce à une identifiant, figure 3.

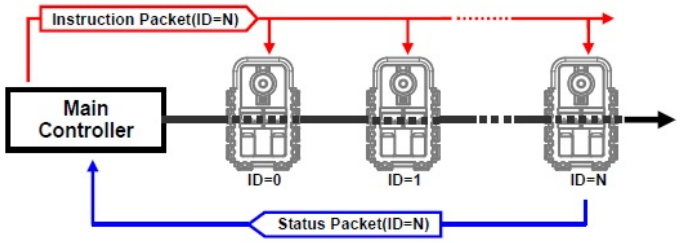


FIGURE 3 – Servomoteurs en cascade

Les principales caractéristiques de ce moteur sont les suivantes :
— Alimentation : 9 à 12 Vcc avec 11.1 v recommandée
— Courant de blocage : 1.5 A
— Identifiant : de 0 à 253
— Poids : 54.6
— Résolution :0.29°
— Dimensions 32 x 50.1 x 40 mm

Pour le fonctionnement du servomoteur, il est nécessaire de spécifier l'instruction, figure 4, et les éléments pour que l'AX12 puisse identifier et exécuter l'action, il faut noter que toutes les trames doivent commencer par 0xFF 0XFF.

Instruction	Function	Value	Number of Parameter
PING	No action. Used for obtaining a Status Packet	0x01	0
READ DATA	Reading values in the Control Table	0x02	2
WRITE DATA	Writing values to the Control Table	0x03	2 ~
REG WRITE	Similar to WRITE_DATA, but stays in standby mode until the ACION instruction is given	0x04	2 ~
ACTION	Triggers the action registered by the REG_WRITE instruction	0x05	0
RESET	Changes the control table values of the Dynamixel actuator to the Factory Default Value settings	0x06	0
SYNC WRITE	Used for controlling many Dynamixel actuators at the same time	0x83	4~

FIGURE 4 – Instructions du servomoteur

Il existe deux façon de commander le moteur : "Write_data" qui exécute l'action juste après avoir reçue l'instruction et "Reg_data" qui attend une autre instruction pour exécuter l'action. Ces deux modes commandent le moteur par position et non par couple.

Pour envoyer une commande, il faut spécifier quelques paramètres :

— L'ID du servomoteur.
— La longueur N mais on envoie N+3.
— L'instruction, figure 4.
— Les paramètres.
— Checksum pour le contrôle d'erreurs.
Un exemple de commande en "Write_data" est dans la figure 5 :

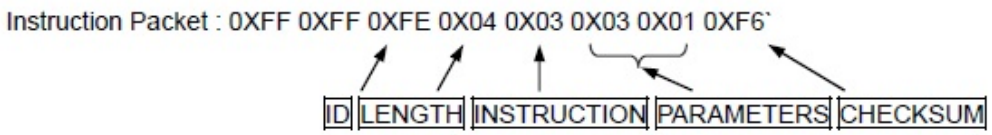


FIGURE 5 – Exemple pour envoyer une instruction au moteur

Pour récupérer les informations des moteurs comme la position, vitesse , couple ou tension ; on envoie une trame spécifiant l'instruction et l'adresse du type d'information qu'on veut :
— Vitesse : 0X20
— Position : 0X1E
— Couple : 0X18
— Tension : 0X2A
— Température : 0X2B

2 Carte de développement NUCLEO-F746ZG

Ce microcontrôleur commandera les moteurs avec une communication série asynchrone et nous permettra d'ajouter des éléments pour améliorer l'apprentissage par renforcement tels que des capteurs comme une gyroscope pour stabiliser la base de notre robot.

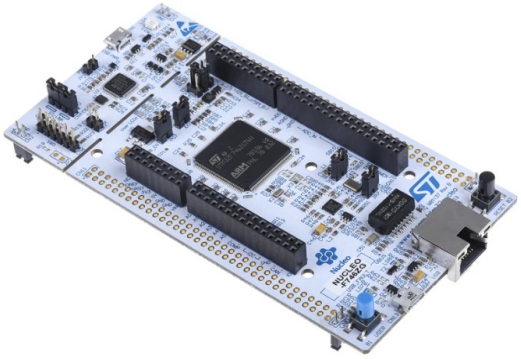


FIGURE 6 – Carte NUCLE F746ZG

3 Jetson nano

Cette carte est conçue par NVIDIA et permet le développement de systèmes AI compacts à faible coût et à faible consommation. Dans la gamme de produits de cette entreprise, il existe d'autres cartes ,plus puissantes mais plus chères, qui sont de très hautes performances pour des applications plus complexes.

FIGURE 7 – Jetson Nano

2 Construction du fichier URDF

La simulation est une étape très importante du projet car elle nous renseigne sur la viabilité du projet. Cependant, une étape préalable à cela est la modélisation numérique du robot avec toutes ses caractéristiques physiques pour se rapprocher du modèle réel, pour cela nous avons récupéré les ressources numériques de l'entreprise qui a construit le robot MIMI qui contient la version numérique du robot dans le programme Solidworks. Cette version est parfaite et constitue un grand pas en avant dans le projet, car elle nous fait gagner du temps et contient les caractéristiques physiques du robot.

SolidWorks est un logiciel propriétaire de conception assistée par ordinateur 3D fonctionnant sous Windows qui contient de nombreux outils de modélisation de robots. Un outil très pratique est "Exporter en URDF" qui doit être activé avant utilisation.

1 Fichier URDF

Un fichier URDF (Universal Robotic Description Format) est un format de description universel qui est utilisé un ROS (Robot Operating System) et il est écrit sur un format XML. Ce fichier décrit tous les éléments d'un robot.

Il existe un tutoriel fait par ROS [1] où nous pouvons apprendre les bases pour comprendre et créer notre propre fichier URDF. Nous devons d'abord comprendre que la structure de ce fichier a une structure arborescente comme l'image 8 qui décrit des solides qui sont reliés par des liaison qui seront spécifiées, dans notre robot ce liaisons sont pivots.

Pour pouvoir place et fixé les solides et l'axe de rotation des liaisons, des repères sont créés pour chaque solide.

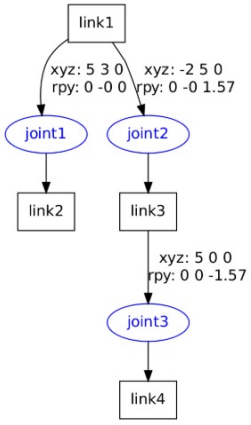


FIGURE 8 – URDF

Pour définir des solides, nous pouvons utiliser des fonctions pour créer des solides simples comme des sphères, des parallélépipèdes et des cylindres, mais la plupart des projets utilisent des solides complexes. Pour cela, on utilise des fichiers STL (STereo-Lithography) qui ne décrivent que la géométrie de surface d'un objet un 3D.

Dans le paragraphe précédent, nous avons spécifié l'élément visuel qui définit à quoi ressemble le robot, mais pour faire une simulation, la détection des collisions est très importante car elle définit la forme des objets. Nous pouvons utiliser des fonctions pour définir des formes simples ou un fichier STL de la même manière que pour l'élément visuel mais il faut prendre en compte que la détection de collision est très complexe en termes de calcul que pour de géométries simples ce qui conduit à un traitement plus lent.

2 L'URDF dans SolidWorks

Comme vu précédemment, pour créer un fichier URDF, il est nécessaire des créer les fichiers STL, définir les liaison, les repères et ajouter les caractéristiques physiques de chaque objet qui compose le robot mais l'assemblage du robot, obtenue des ressources numériques du constructeur du robot, a ces caractéristiques physiques et les objets sont totalement définis. Cependant, pour que l'outil "Export as URDF" fonctionne, il faut que toutes les solides de l'assemblage soient bien définies mais ce n'était pas le cas. Or, nous avions toutes les parties du robot bien définis et il suffisait de faire l'assemblage avec ces pièces, figure 9 .

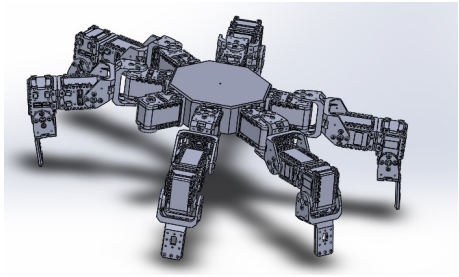


FIGURE 9 – Robot MMIMI modélisé en SolidWorks

Comme nous voulions d'abord appliquer l'IA à une patte avant qu'à l'hexapode, nous avions besoin de deux fichiers urdf pour ces deux assemblages mais le résultat était un une modélisation avec des pièces mal définies et décalées en rotation et en translation. En essayant de résoudre ce problème, nous avons réalisé que cet outil de Solidsworks avait des bogues et que beaucoup de gens ne pouvaient pas obtenir le fichier urdf. Cependant, notre seul problème était le déplacement des pièces et la solution était d'éditer les positions pour obtenir un bon résultat.

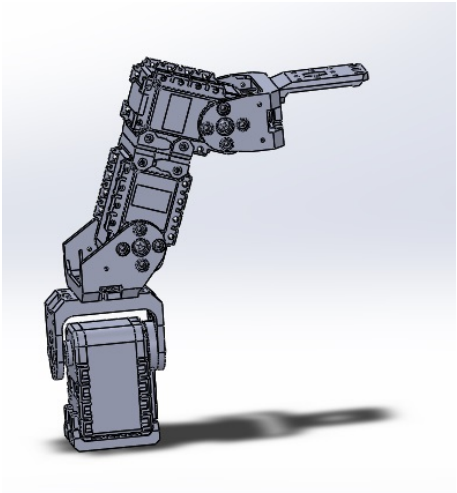


FIGURE 10 – Patte du robot modélisé en SolidWorks

Ci-dessous, nous montrons une partie du fichier urdf de la patte où une pièce est définie.

```
<robot
  name="URDF_Patte_F1">
  <link
    name="base_link">
    <inertial>
      <origin
        xyz="-2.9577E-05 0.026911 0.00093243"
        rpy="0 0 0" />
      <mass
        value="0.045999" />
      <inertia
        ixx="2.9262E-06"
        ixy="6.1007E-10"
        ixz="1.5001E-10"
        iyy="5.2146E-07"
        iyz="-3.8096E-07"
        izz="2.4785E-06" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
```

```
      <mesh
        filename="package://URDF_Patte_F1/meshes/base_link.STL" />
    </geometry>
    <material
      name="">
      <color
        rgba="0.79216 0.81961 0.93333 1" />
    </material>
  </visual>
  <collision>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://URDF_Patte_F1/meshes/base_link.STL" />
    </geometry>
  </collision>
</link>
```

3 Apprentissage par renforcement

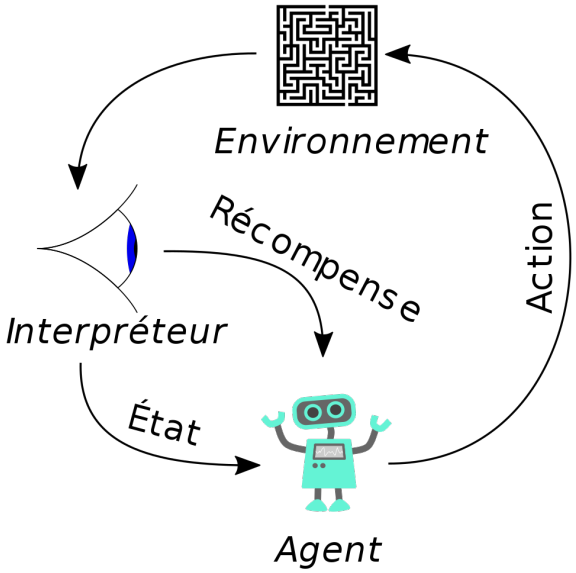


FIGURE 11 – Principe de l'apprentissage par renforcement

En *apprentissage par renforcement*, on considère un **agent** et un **environnement**. Le but est d'apprendre à l'agent, par exemple un robot, à effectuer certaines tâches. L'idée est représentée FIGURE 11. L'agent se trouve dans un certain **état** et décide, en fonction de cet état courant, d'effectuer une **action**. Il agit sur son environnement qui lui renvoie alors un nouvel état et une **récompense** via un **observateur** (ou *interpréteur*). L'apprentissage consiste à déterminer une **politique**, *ie* une fonction qui à chaque état de l'agent associe l'action à effectuer, qui maximise la récompense au cours du temps, ou de manière plus générale une quantité qui en dépend. Une telle politique est dite *optimale*. On espère ainsi qu'à la fin de la phase d'apprentissage, l'agent réalise bel et bien la tâche qu'il était censé apprendre.

Par ailleurs, on observe, FIGURE 11, que *l'apprentissage dépend explicitement de la récompense reçue par l'agent et de ses observations au cours des différentes itérations.* Ainsi, l'attribution de la récompense doit être gérée de sorte qu'une récompense favorable soit renvoyée à l'agent lorsqu'il effectue une "bonne action" et une récompense défavorable lorsqu'il effectue une action inutile, voire contre-productive.

1 Formalisme

L'approche classique pour modéliser mathématiquement la situation est celle des **processus de décision markoviens** (*Markov Decision Process [MDP]*) [2]. Il s'agit d'un quadruplet $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$, avec respectivement $\mathcal{S}$ l'ensemble des états possibles, $\mathcal{A}$ l'ensemble des actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ une *fonction de transition* qui à chaque triplet $(s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ associe la probabilité $\mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a)$ que l'agent se trouve dans l'état s' à l'instant $t + 1$ sachant qu'à l'instant t il se trouve dans l'état s et entreprend l'action a. $(S_t)_{t\in\mathbb{N}}$ et $(A_t)_{t\in\mathbb{N}}$ forment des suites de variables aléatoires, respectivement à valeurs dans $\mathcal{S}$ et $\mathcal{A}$. Enfin, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ est la **fonction de récompense**, qui à chaque transition associe une récompense scalaire réelle. Notons que dans le cas général d'apprentissage par renforcement, le MDP est sans mémoire et vérifie donc la **propriété de Markov**.

On appelle **politique** de l'agent, la fonction qui permet à ce dernier de décider à chaque état quelle action choisir. Cette fonction peut se définir comme $\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$, qui associe à chaque couple $(a, s) \in \mathcal{A} \times \mathcal{S}$ la probabilité qu'a l'agent de choisir l'action a à partir de l'état s : $\mathbb{P}(A_t = a | S_t = s)$.

Le but de l'apprentissage est de déterminer la fonction π^* qui maximise un certain **retour** G, une quantité qui dépend des récompenses obtenues par l'agent lors du parcours du *MDP*. Ainsi, pendant la phase d'apprentissage, les probabilités associées au différentes transitions sont modifiées de sorte à maximiser ce retour. Les transitions amenant à de faibles valeurs de retour voient leur probabilité associée diminuer, celles menant à des valeurs maximales de G voient leur probabilité augmenter, d'où le terme *renforcement*. Les transitions menant à des valeurs de retour maximales sont "renforcées", les autres "inhibées".

Dans la plupart des algorithmes d'apprentissage par renforcement, notamment dans le cas du plus célèbre d'entre-eux qu'est le Q-learning, le retour G_t obtenu depuis un temps t prend la forme suivante :

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1}$$

Avec γ un réel compris entre 0 et 1 qui quantifie la prise en compte les récompenses futures. En effet, si $\gamma = 0$, alors les récompenses futures ne sont pas prises en compte. En outre, ce facteur permet la convergence de la somme précédente.

Ou de manière récursive :

$$G_t = R_t + \gamma G_{t+1} \tag{2}$$

Cependant, on remarque que G_t est une variable aléatoire. On ne va donc pas chercher à maximiser directement G_t mais plutôt le *retour espéré*. C'est ce que l'on appelle la **fonction de valeur** (*value function* en anglais) [2] :

$$v_\pi(s, a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] \tag{3}$$

La valeur $v_\pi(s, a)$ quantifie le retour que l'agent peut espérer obtenir en choisissant l'action a quand il se trouve dans l'état s, avec une politique π.

2 Application à la robotique

En robotique, *l'état* correspond aux *variables d'état du système* en automatique : positions-vitesses-couples au niveau des moteurs, position du robot dans un certain repère, assiette (dans le cas d'un drone par exemple), etc. Une *action* correspond à une *commande* appliquée aux moteurs, en position ou en vitesse par exemple. Se posent alors une série de problèmes concernant l'implémentation de l'apprentissage [3]. Tout d'abord, l'espace des états accessibles par le robot est continu. Il en est de même pour l'espace des actions. En effet, les systèmes physiques, comme les moteurs, sont des systèmes continus. L'état du robot est ainsi obtenu par la mesure de variables continues. De même, les actions correspondent à des tensions envoyées aux moteurs. En outre, ces espaces sont en général d'assez grande dimension. Dans le cas d'un bras robotisé à n joints dont l'état correspond aux positions des n moteurs au niveau des liaisons mécaniques, on a un espace d'état de n dimensions. Ainsi, même en discrétisant ces espaces, le nombre d'états et d'actions possibles est énorme, ce qui entrave la phase d'exploration lors de l'apprentissage. Bien entendu, la *discrétisation*, nécessaire pour effectuer des calculs par ordinateurs, est elle même un défi technique en ce qu'elle ne doit pas perturber l'apprentissage. Par ailleurs, du fait du grand nombre d'états/actions possibles et pour des raisons de sécurité et d'usure des composants [3], il apparaît nécessaire d'effectuer une partie de l'apprentissage en simulation et non en réel. Il apparaît alors un nouveau problème, le *reality gap*, c'est-à-dire l'écart entre la simulation et la réalité, qui provoque des comportements différents. En effet, la simulation suppose de nombreuses hypothèses simplificatrices via la modélisation du robot.

Pour pallier à ce problème, il est possible d'effectuer plusieurs apprentissages avec différentes valeurs pour les paramètres du robot [4], comme la masse de certaines pièces, la tension aux bornes de la batterie, etc. Cela permet de garantir une certaine **robustesse**. Concrètement, on assimile ces différents paramètres à des variables aléatoires suivant une loi gaussienne. On évite ainsi de forts écarts entre le comportement de l'agent en simulation et en réel car on évite au maximum le fait pour l'agent de tomber sur des états qu'il n'aura pas découvert au préalable en simulation. On peut également soumettre le robot à des **perturbations** en simulation de sorte à éviter qu'un léger changement d'état en réel provoque des décisions incohérentes de la part du robot. On améliore encore une fois la robustesse. Toutefois, cela n'empêche pas les problèmes de modélisation. En effet, *Coumans* et son équipe [4] ont maints fois dû adapter leur simulation au comportement réel du robot.

Un autre problème majeur apparaît avec l'application de l'apprentissage par renforcement à la robotique. Il s'agit de l'attribution des récompenses. C'est le problème du ***reward shapping***. On ne peut pas, comme un jeu de morpion, appliquer le principe du *task achievement*, *ie* donner une très bonne récompense une fois un succès atteint. En effet, le nombre d'états possibles à explorer est en général beaucoup trop grand pour un robot. Dans le cas d'un bras robotisé devant jouer au ping pong, cela est tout simplement abérant [3]. Dans le cas du morpion cela est possible car le nombre d'états possibles est fini, et qu'à partir d'un état, on ne peut effectuer qu'assez peu d'actions. Il est également d'autant plus difficile pour un concepteur d'impulser un comportement désiré via la récompense, car le comportement du robot lors de l'apprentissage et après, du fait des nombreuses configurations géométriques possibles, est assez imprévisible. Ainsi, un comportement

qui maximise la récompense peut ne pas correspondre à un comportement désiré par le concepteur [3]. Enfin, un compromis doit être trouvé entre complexité du problème et complexité de la fonction de récompense.

3 Choix du modèle

3.1 Environnement

L'environnement modélise la manière dont l'agent interagit avec le monde qui l'entoure. Lorsque **l'agent**, alors dans un *état* S_t, effectue une action A_t, l'environnement lui renvoie son *nouvel état* S_{t+1} et une *récompense* R_{t+1} (FIGURE 12).

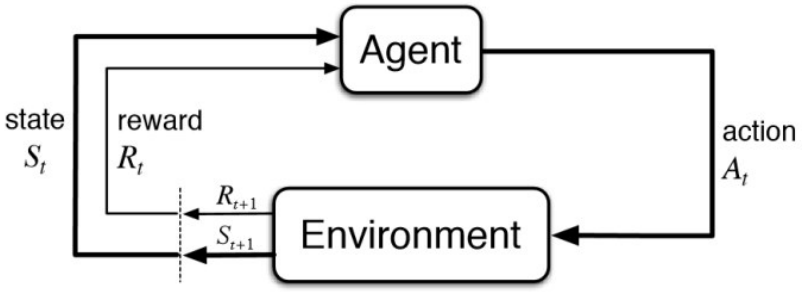


FIGURE 12 – Environnement

Dans notre cas, celui de l'hexapode, **l'espace des observations est continu**. Cet espace comporte autant de dimensions que de variables d'état. Ces variables sont les positions angulaires au niveau des moteurs, l'inclinaison de la plate-forme et sa hauteur. En réel, l'état est obtenu via des capteurs, d'où le terme *observation*. Les servomoteurs *AX12* peuvent communiquer leur position et leur vitesse à un microcontrôleur. Ce même microcontrôleur, via une *centrale inertielle* peut déterminer l'inclinaison de la plate-forme. Un *télémètre* peut servir à mesurer la hauteur. En simulations, ces données sont obtenues via les calculs d'un *simulateur physique*, PyBullet dans notre cas (FIGURE 13).

L'espace des actions est en général continu [1]. Ces actions correspondent aux commandes en position envoyées aux moteurs à chaque époque t. La commande est effectuée à l'aide d'un **contrôle PD** (proportionnel-dérivé) avec une commande en vitesse constante. Seule la position correspond à l'action. En effet, les moteurs *AX12* permettent de rejoindre une position à vitesse à peu près constante.

Conformément à l'article [4], il est nécessaire d'avoir les dimensions les plus faibles possibles pour les espaces d'actions et d'états. En effet, cela permet d'éviter les problèmes de sur-apprentissage et d'overfitting. Ainsi, pour notre espace d'actions, on se limite aux positions. Dans le cadre d'une poursuite de ce travail de recherche, nous pourrions inclure également les vitesses mais il faudrait alors bénéficier d'un calculateur et probablement trouver une façon de réduire notre modèle.

1. Nous examinerons les différents cas dans la partie implémentation

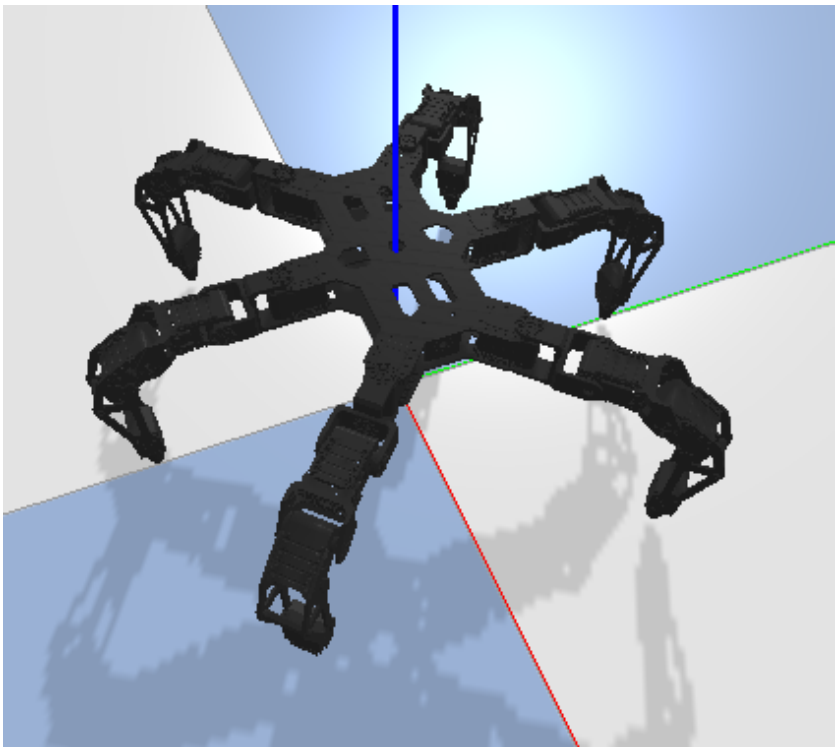


FIGURE 13 – Modèle de l'hexapode Phamtomx dans notre simulation sous PyBullet

Dans un premier temps, on cherche à obtenir un robot qui marche le plus loin possible dans une direction. Par conséquent, la **récompense** doit augmenter avec la distance parcourue. Une première approche naïve consiste à directement utiliser la distance parcourue comme récompense : $R_t = X_t - X_{t-1}$. L'approche proposée par l'article [4] prend également en compte l'*énergie consommée* :

$$R_t = X_t - X_{t-1} - w.|\mathbf{T}_t.\mathbf{\Omega}_t| \tag{4}$$

avec $\mathbf{T}_t$ le vecteur formé à partir des couples à l'instant t au niveau de chaque moteur et $\mathbf{\Omega}_t$ celui formé à partir des vitesses angulaires à l'instant t. Quand à w, il s'agit d'un **hyperparamètre** du modèle. C'est un réel constant qui quantifie la prise en compte de la puissance mécanique dans la récompense.

De manière générale, la récompense permet d'impulser un certain comportement en ce qu'elle guide l'apprentissage. C'est, en effet, elle qui indique à l'algorithme d'apprentissage comment modifier les paramètres de la politique. Ainsi, on modifiera la fonction de récompense en fonction des résultats obtenus à la fin de chaque apprentissage pour forcer le robot à adopter le comportement que l'on souhaite.

3.2 Algorithme d'apprentissage

Nous avons notre *environnement*. Nous l'avons vu, ce dernier permet de déterminer le nouvel état du robot à partir de l'état précédent et de l'action entreprise. Reste maintenant à déterminer la **politique** π que va suivre le robot pour décider à partir de son état courant S_t, l'action A_t qu'il va entreprendre. C'est l'objectif de l'algorithme d'apprentissage.

Tout d'abord, nous avons des espaces continus d'états et d'actions. Par conséquent, toutes les approches discrètes d'apprentissage par renforcement ne peuvent s'appliquer. Nous allons donc utiliser un **réseau de neurones**, qui permet de manipuler directement des quantités continues. De plus, l'avantage d'un réseau de neurones est sa capacité de **généralisation**. En effet, lors de la phase d'apprentissage, le réseau est entraîné à partir de données, états et récompenses obtenues au cours de chaque épisode. Une fois l'apprentissage réalisé, il ne tombera pas sur les mêmes états lors de son utilisation [3]. Le réseau va alors effectuer, de fait, une interpolation. Les états proches de ceux qu'il a rencontré lors de l'apprentissage donneront en sortie une action semblable.

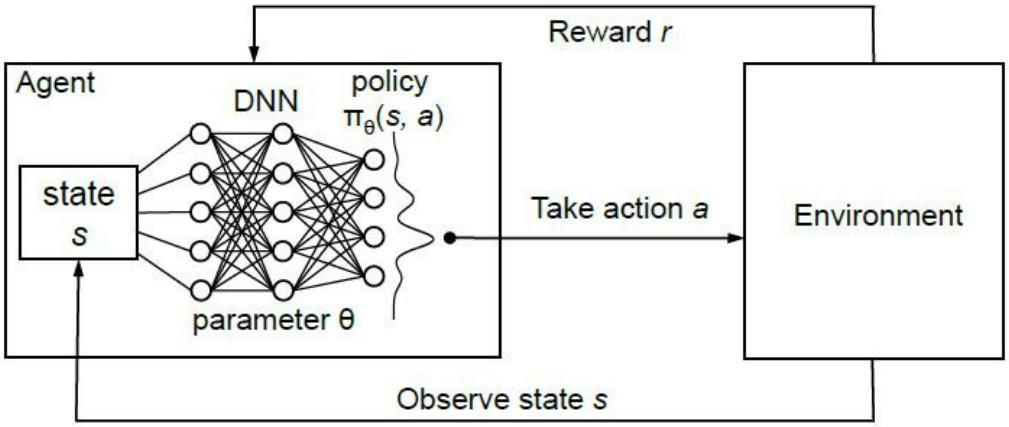


FIGURE 14 – Architecture classique en Deep Reinforcement Learning

Les paramètres θ du réseau, *ie* de la **politique** π, sont les poids au niveau des synapses du réseau. Le but de l'algorithme d'apprentissage est de déterminer les paramètres θ optimaux qui maximisent le retour espéré (Cf SECTION 1) via les récompenses. Pour notre architecture, nous utilisons un réseau de neurones à plusieurs couches (MultiLayer Policy : MLP). Cela sera détaillé dans la partie implémentation.

Nous avons fait le choix de l'algorithme **PPO (Proximal Policy Optimization)** [5]. Le choix s'est porté sur cet algorithme car il offrait les meilleures performances pour des environnements continus (FIGURE 15).

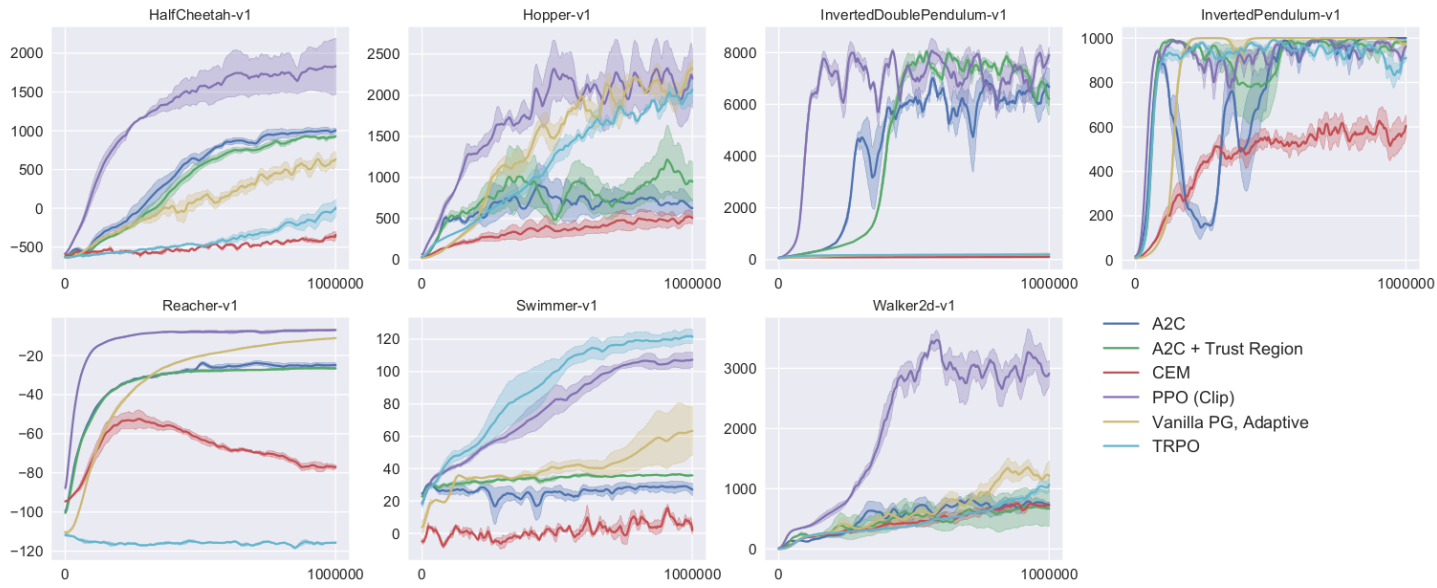


FIGURE 15 – Performances de différents algorithmes de Deep Reinforcement Learning
Ici sont représentées les récompenses obtenues pour chaque époque, en gras la récompense moyenne et en plus transparent la plage de variation de la récompense pour différents apprentissages.

Fonctionnement de PPO

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

FIGURE 16 – Algorithme PPO en langage naturel

L'algorithme peut effectuer du parallélisme, *ie* mettre à jour la politique à l'aide de plusieurs agents agissant en parallèle, ce que nous n'avons pas essayé.

A chaque itération, l'algorithme fait évoluer l'agent sur un certain nombre T d'époques. En clair, l'agent effectue T actions déterminées via la politique courante π_{old}. A chaque époque $t \in \{1...T\}$ est associée un avantage $\widehat{A}_t$. Cet avantage est un scalaire qui prend en compte les récompenses obtenues. Ces avantages permettent de calculer une quantité L, qui dépend également de la politique et que nous expliciterons plus bas, qui doit être maximisée. Les paramètres θ du réseau sont alors mis à jour à l'aide d'une ascension de gradient : $\theta \leftarrow \theta_{old} + \alpha \nabla_\theta L$.

En effet, l'algorithme consiste en une optimisation de la politique par ascension de gradient : **policy gradient optimization** [3] [5].

L'algorithme est basé sur une approche semblable à celle de **TRPO (Trust Region Policy Optimization)**, un autre algorithme d'apprentissage [5] [6]. Il s'agit de faire en sorte que la nouvelle politique obtenue par la mise à jour des paramètres θ ne diffère pas trop de l'ancienne.

Dans la plupart des approches d'optimisation par ascension de gradient, la fonction L à optimiser est définie de la manière suivante [5] :

$$L^{PG}(\theta) = \widehat{\mathbb{E}}_t \left[\log \pi_\theta(a_t|s_t) \widehat{A}_t \right] \tag{5}$$

La démonstration de cette formule figure dans le cours *Policy Gradient* de Sergei Levine [7].

Dans la pratique, *ie* dans un algorithme, l'opérateur $\widehat{\mathbb{E}}_t$ est approché par une moyenne des T termes $\log \pi_\theta(a_1|s_1)\widehat{A}_1 ..... \log \pi_\theta(a_T|s_T)\widehat{A}_T$ [5].

Cependant, dans les algorithmes PPO et TRPO, ce n'est pas cette quantité qui est utilisée. Le terme en logarithme est remplacé par le terme $r_t(\theta)$:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{6}$$

Ainsi :

$$L^{TRPO}(\theta) = \widehat{\mathbb{E}}_t \left[r_t(\theta) \widehat{A}_t \right] = \widehat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \widehat{A}_t \right] \tag{7}$$

D'après cette formule, lorsque $\widehat{A}_t$ est positif, c'est à dire que les actions de l'agent apportent de bonnes récompenses, la mise à jour des paramètres θ renforce les transitions qui ont vont dans ce sens. Comme on l'a vu précédemment ces actions seront désormais plus probables. En revanche, lorsque $\widehat{A}_t$ est négatif, les transitions sont inhibées.

La principale modification apportée à TRPO par PPO est le **clipping**. Il s'agit d'éviter les trop fortes valeurs de gradient et ainsi éviter que la nouvelle politique s'éloigne trop de l'ancienne :

$$L^{CLIP}(\theta) = \widehat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\widehat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\widehat{A}_t \right) \right] \tag{8}$$

Avec ϵ un paramètre entre 0 et 1, en général égal à 0.2 [5].

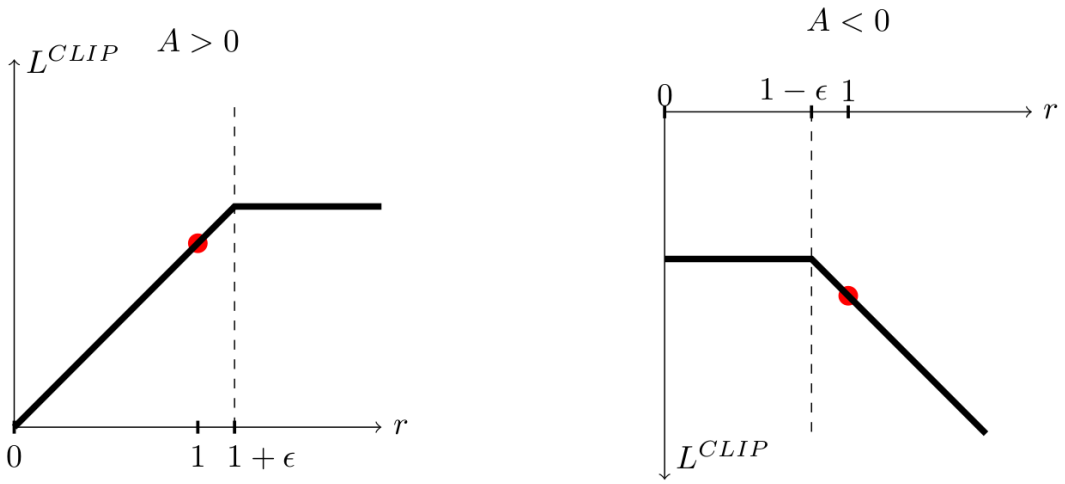


FIGURE 17 – Illustration du phénomène de clipping

4 Implémentation

L'ensemble de la simulation a été réalisée en Python.

4.1 Environnement

Open AI Gym

L'environnement en lui-même ; *ie* la détermination de l'état courant, celle de l'état suivant et de la récompense à partir de l'action entreprise ; est implémenté à l'aide de la bibliothèque **Open AI Gym**.

La structure est la suivante :

exemple.py

```
class Exemple(gym.Env):

    """
    Hérite de la classe d'environnements Gym
    """

    def __init__(self, args):
    """
    Constructeur

    Définition de l'espace des actions
    et de l'espace des états (ou observations)
    """

    def reset(self):
    """
    Réinitialisation de l'environnement :
    retour du robot à sa position initialepar exemple
    """

    def step(self, action):
    """
    revoie un tuple : (etat, reward, done, info)

    - etat : nouvel état
    - reward : récompense
    - done : épisode terminé (booléen)
    - info : dictionnaire contenant des informations complémentaires
    """

    def render(self):
    """
    rendu graphique
    """
```

A ces méthodes peuvent bien entendu s'en ajouter d'autres en fonction des nécessités.

On retrouve dans l'architecture de la classe le formalisme de l'apprentissage par

renforcement exposé Figure 12.

L'idée d'OpenAI Gym est d'implémenter un environnement sur lequel on peut alors appliquer n'importe quel algorithme d'apprentissage. Il s'agit en quelque sorte d'une couche d'abstraction.

Enfin, Gym permet de coder 4 formes d'espaces : *Discrete* (variable entière au nombre fini de valeurs), *MultiDiscrete* (la même chose en plusieurs dimensions), *Float* (variable flottante bornée), *Box* (la même chose en plusieurs dimensions).

Nous avons donc utilisé des espaces de type *Box*.

PyBullet

Or pour déterminer l'état de notre robot et qu'il effectue des actions, un simulateur physique est nécessaire. On utilise donc **PyBullet** au sein de notre environnement calculer toutes les variables de l'environnement : état, action, récompenses.

On peut ainsi traduire les actions en commandes moteur, *PyBullet* permettant l'implémentation d'une correction PD pour les moteurs.

On peut également obtenir toutes les informations nécessaires au niveau des moteurs : position, vitesse, couple, etc. On peut également effectuer des calculs d'angles d'Euler à partir des quaternions pour la base.

4.2 Algorithme d'apprentissage

Pour l'apprentissage nous utilisons la bibliothèque *stable-baselines*. Cette bibliothèque regroupe de nombreux algorithmes pré-implémentés, notamment PPO.

Concernant l'architecture de notre réseau de neurones, nous avons utilisé *MlpPolicy* dans *stable-baselines* qui comporte par défaut deux couches de 64 neurones. Il aurait été intéressant de jouer d'apprendre à jouer sur l'architecture du réseau pour l'apprentissage mais nous avons manqué de temps à y consacrer et n'avons d'ailleurs pas effectuer beaucoup de recherches sur le sujet. C'est une piste de recherche pour de futurs travaux.

Toutefois, cela ne nous a pas empêché d'obtenir des résultats intéressant pour l'apprentissage d'une tâche simple sur une patte.

5 Résultats

Les codes des différentes simulations figurent en annexe.

La création du fichier *URDF* de notre robot ayant pris un certain temps, nous avons en parallèle effectué des tests à partir d'un exemple existant trouvé sur GitHub : *Phantomx*.

Nous avons d'abord effectué des tests sous PyBullet. Puis, après nous être exercé sur Gym via des exemples simples de jeu que nous avons nous mêmes créés, comme l'environnement *Blob* joint en annexe, nous avons effectué une première implémentation naïve d'environnement pour l'hexapode. En effet, l'espace des actions était discret, inspiré des requêtes AX12.

Puis nous avons implémenté un environnement continu et fonctionnel. Malheuresement, l'apprentissage prend beaucoup trop de temps sur une machine conventionnelle. Nous avons donc réalisé un environnement simple à partir d'une patte de notre robot :

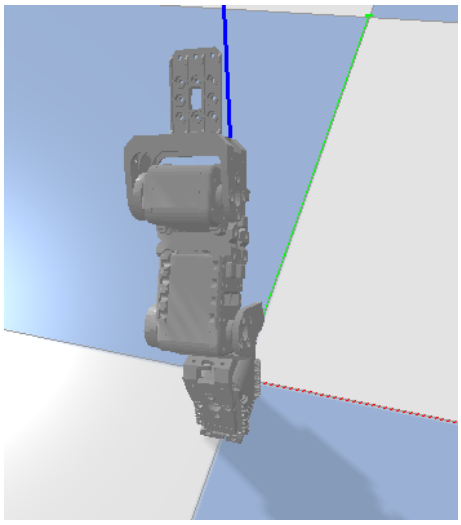


FIGURE 18 – Aperçu de la patte

Le but était d'apprendre à la patte à maximiser sa hauteur. Pour cela, nous avons considéré comme état sa hauteur et les positions des moteurs, comme action des commandes en position sur les moteurs.

Pour différentes fonctions de récompense, nous avons pu obtenir des résultats intéressants :

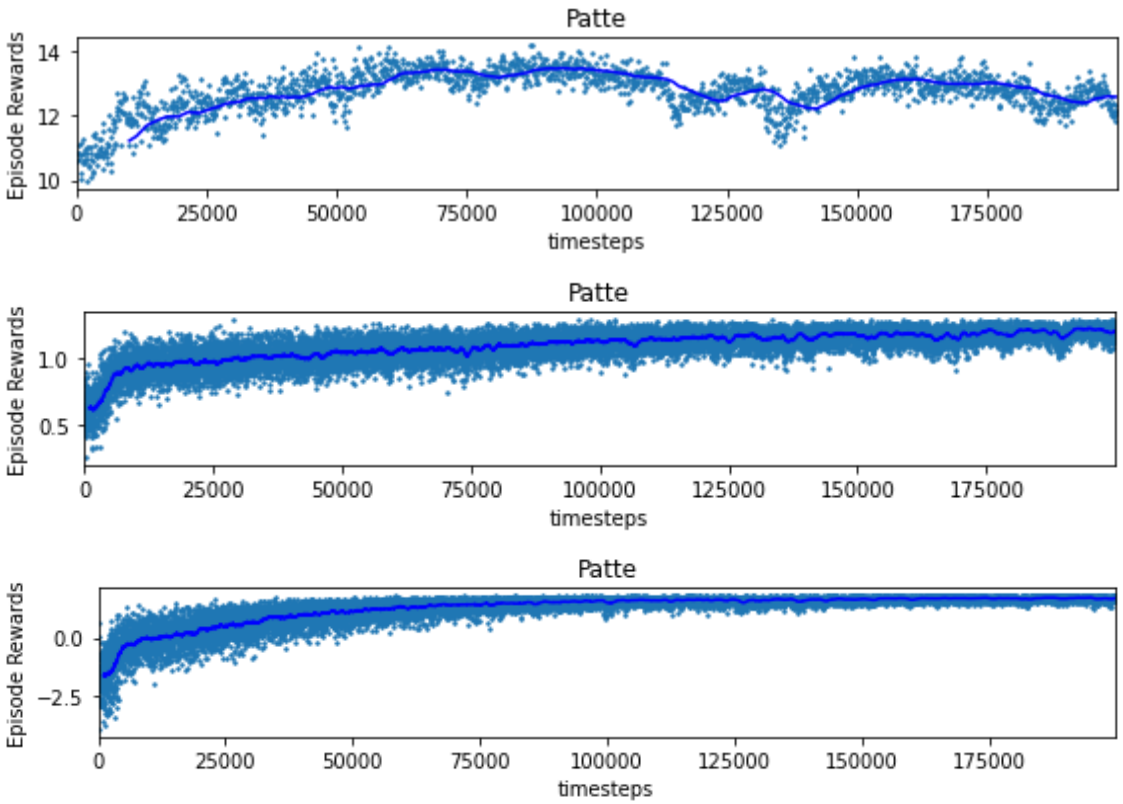


FIGURE 19 – Récompense pour chaque époque : moyenne en gras

Dans le premier cas, la récompense est directement la hauteur de la patte : $r_t = h_t$. On observe que la récompense atteint un maximum avant de descendre. En outre, bien que la patte tend à aller autour de la position maximisant la hauteur, celle-ci est souvent loin de l'optimum, comme le montrent la dispersion des points autour de la moyenne.

Dans le second cas, la récompense est de la forme : $r_t = h_t - Kt$, avec $K = 0.01$. On ajoute une pénalité pour le temps que la patte met à trouver une position optimale. On obtient alors une amélioration sur la forme de l'évolution de la récompense mais pas sur la variance (écart à la moyenne).

Dans le dernier cas, on fait dépendre la pénalité du dernier maximum de hauteur atteint : $r_t = h_t + (h_t - max_t h_t)t$. On obtient ainsi une variance qui diminue avec le temps. L'écart entre la position optimale et celle choisie par la patte est alors bien moins grand.

On aurait très bien pu effectuer de la **cinématique inverse** pour résoudre ce problème, ce qui est d'ailleurs bien plus efficace quand on dispose d'une modélisation géométrique de la patte. Cependant, cet exemple simple nous a permis de mettre en évidence les effets du **reward shapping** sur l'apprentissage et ainsi d'obtenir des résultats intéressants à exploiter.

Conclusion et perspectives

En définitive, ce TER fut une bonne introduction aux thématiques d'apprentissage par renforcement appliqué à la robotique. En effet, nous avons pu avoir un aperçu de l'état de l'art et apprendre à maîtriser les outils essentiels à sa mise en œuvre.

Toutefois, le travail de recherche était sans doute trop grand pour une année chargée. Par ailleurs, nous ne disposions pas de calculateur, ce qui a pu nous a empêché de réaliser de véritables tests sur l'hexapode en simulation.

Ainsi, il aurait été intéressant que nous simplifions le problème afin de compresser le temps d'apprentissage. Aussi, peut-être qu'une approche basée sur un modèle nous aurait permis d'aller dans ce sens. Nous aurions pu également effectuer de l'imitation, ce que permet stable-baselines. Il s'agirait d'implémenter une marche manuellement puis d'optimiser cette marche via un apprentissage. Cependant, cette dernière approche aurait introduit un *a priori*. Il y a donc un compromis à effectuer entre l'*a priori* et l'apprentissage.

Enfin, nous envisageons de poursuivre ce travail, au moins en simulation. En cas de succès une approche intéressante serait de poursuivre avec un **apprentissage hiérarchique**. On apprend des routines au robot : avancer, tourner, etc. Puis, on apprend à l'agent à planifier une trajectoire. Les actions de ce "méta-apprentissage" sont alors les routines [8].

Codes

simulations

Environnement Patte

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-



import os

import numpy as np

import matplotlib.pyplot as plt

import gym
from gym import spaces
from gym.utils import seeding

from stable_baselines.common.policies import MlpPolicy
from stable_baselines.common.policies import CnnPolicy # réseaux convolutionnels
from stable_baselines.common.vec_env import DummyVecEnv
from stable_baselines import PPO1
from stable_baselines import PPO2
from stable_baselines import A2C
#from stable_baselines import TRPO
from stable_baselines.common.env_checker import check_env

# pour faire un suivi de l'apprentissage
from stable_baselines import results_plotter
from stable_baselines.bench import Monitor
from stable_baselines.results_plotter import load_results, ts2xy
#from stable_baselines.common.noise import AdaptiveParamNoiseSpec
from stable_baselines.common.callbacks import BaseCallback

# pré-entrainement
from stable_baselines.gail import generate_expert_traj

# modules de simulation physique
import pybullet as p
import pybullet_data

import time

from math import pi



class Patte_env(gym.Env):
    metadata = {
        'render.modes': ['human', 'rgb_array'],
        'video.frames_per_second': 50
    }
```

```
50
51
52
53      def _seed(self, seed=None):
54          self.np_random, seed = seeding.np_random(seed)
55          return [seed]
56
57      def __init__(self, render=True):
58
59          super(Patte_env, self).__init__()
60
61          self.render = render
62
63          if (render):
64              self.physicsClient = p.connect(p.GUI)
65          else:
66              self.physicsClient = p.connect(p.DIRECT)  # non-graphical version
67
68          p.setAdditionalSearchPath(pybullet_data.getDataPath())  # used by loadURDF
69
70          # position initiale de la camera
71          p.resetDebugVisualizerCamera(cameraDistance=0.8, cameraYaw=0, cameraPitch
        =-30, cameraTargetPosition=[0, 0, 0])
72
73          self._seed() # initialisation de la graine (pour l'aléatoire)
74
75          p.resetSimulation()
76          p.setGravity(0, 0, -9.81)  # m/s^2
77
78          self.dt = 0.01 # pas de temps de la simulation (en secondes)
79          p.setTimeStep(self.dt)   # sec
80
81          self.periode_actions = int(5/self.dt)
82
83          self.longueur_episode = 10 # nombre d'étapes dans un épisode
84          self.compteur_etapes = 0
85
86
87          self.plane = p.loadURDF("plane.urdf")
88
89
90          # position de départ de la patte
91          self.cubeStartPos = [0, 0, 0]
92          self.cubeStartOrientation = p.getQuaternionFromEuler([0, 0, 0])
93
94
95          # chargement du robot dans pybullet
96          self.id_patte = p.loadURDF("URDF_Patte_F1.urdf", self.cubeStartPos, self.
        cubeStartOrientation, useFixedBase=1)
97
98
99          # espace des actions : contrôle en position : PD control commande constante
         en vitesse
100         self.action_space = spaces.Box(low=-1, high=1, shape=(3,), dtype=np.float32
        )
101
```

```
102         # espace des observations : hauteur du bout de la patte, positions des moteurs
103         self.observation_space = spaces.Box(low=-1, high=1, shape=(4,), dtype=np.float32)
104 
105         # v3 : on conserve le dernier max atteint -> Cf fonction de récompense
106         self.max_hauteur = 0
107 
108 
109     def reset(self):
110 
111         self.compteur_etapes = 0
112         self.max_hauteur = 0
113 
114         for id_moteur in range(3):
115             p.setJointMotorControl(self.id_patte, id_moteur, p.POSITION_CONTROL, pi/4)
116             p.setJointMotorControl(self.id_patte, id_moteur, p.VELOCITY_CONTROL, 0.0)
117 
118 
119         #p.resetSimulation()
120 
121         # on replace le robot à sa position initiale
122         p.resetBasePositionAndOrientation(
123             self.id_patte,
124             posObj=self.cubeStartPos,
125             ornObj=self.cubeStartOrientation
126         )
127 
128         return self.deter_etat()
129 
130 
131     def deter_etat(self):
132 
133         """
134         Détermine l'état de l'agent :
135             * Hauteur du bout de la patte
136             * Position de chaque moteur
137         """
138 
139         return np.array([p.getLinkState(self.id_patte, 2)[0][2],   # z bout de patte
140                          p.getJointState(self.id_patte, 0, self.physicsClient)[0],  # orientation base
141                          p.getJointState(self.id_patte, 1, self.physicsClient)[0],
142                          p.getJointState(self.id_patte, 2, self.physicsClient)[0]], dtype=np.float32)
143 
144 
145     def action_moteurs(self, action):
146 
147         """
148         Traduction des actions en commandes moteur
149         """
150 
```

```
151         for id_moteur in range(3):
152             p.setJointMotorControl2(self.id_patte, id_moteur, p.POSITION_CONTROL,
    targetPosition=pi/3*action[id_moteur]-0.333, targetVelocity=1.0, positionGain
    =0.03, velocityGain=1.0)
153
154
155     def step(self, action):
156
157         """
158         L'agent (la patte) effectue une action.
159         L'environnement lui renvoie son nouvel état et sa récompense.
160         Le booléen "done" permet de déterminer si l'épisode est terminé ou non.
161         """
162
163         self.action_moteurs(action)
164
165         for _ in range(self.periode_actions):
166
167             p.stepSimulation()
168             if self.render: time.sleep(self.dt)
169
170         hauteur = p.getLinkState(self.id_patte, 2)[0][2]
171         if hauteur > self.max_hauteur: self.max_hauteur = hauteur
172
173         # doit rejoindre la position la plus haute le plus vite possible
174         # v3
175         reward = hauteur + (hauteur - self.max_hauteur)*self.compteur_etapes
176
177         self.compteur_etapes += 1
178
179         obs = self.deter_etat()
180
181         done = self.compteur_etapes > self.longueur_episode
182
183         return obs, reward, done, {}
184
185
186     def render(self, mode='human', close=False):
187         pass
188
189     def fermer(self):
190         p.disconnect()
191
192
193
194 ################################### CALLBACK ###########################################
195
196
197
198 class SaveOnBestTrainingRewardCallback(BaseCallback):
199     """
200     Callback for saving a model (the check is done every ``check_freq`` steps)
201     based on the training reward (in practice, we recommend using ``EvalCallback``)
    .
202
203     :param check_freq: (int)
```

```
204     :param log_dir: (str) Path to the folder where the model will be saved.
205       It must contains the file created by the ``Monitor`` wrapper.
206     :param verbose: (int)
207     """
208     def __init__(self, check_freq: int, log_dir: str, verbose=1):
209         super(SaveOnBestTrainingRewardCallback, self).__init__(verbose)
210         self.check_freq = check_freq
211         self.log_dir = log_dir
212         self.save_path = os.path.join(log_dir, 'best_model')
213         self.best_mean_reward = -np.inf
214 
215     def _init_callback(self) -> None:
216         # Create folder if needed
217         if self.save_path is not None:
218             os.makedirs(self.save_path, exist_ok=True)
219 
220     def _on_step(self) -> bool:
221         if self.n_calls % self.check_freq == 0:
222 
223           # Retrieve training reward
224           x, y = ts2xy(load_results(self.log_dir), 'timesteps')
225           if len(x) > 0:
226               # Mean training reward over the last 100 episodes
227               mean_reward = np.mean(y[-100:])
228               if self.verbose > 0:
229                 print("Num timesteps: {}".format(self.num_timesteps))
230                 print("Best mean reward: {:.2f} - Last mean reward per episode:
    {:.2f}".format(self.best_mean_reward, mean_reward))
231 
232               # New best model, you could save the agent here
233               if mean_reward > self.best_mean_reward:
234                   self.best_mean_reward = mean_reward
235                   # Example for saving best model
236                   if self.verbose > 0:
237                     print("Saving new best model to {}".format(self.save_path))
238                   self.model.save(self.save_path)
239 
240         return True
241 
242 
243 ################################## APPRENTISSAGE  ####################################
244 
245 
246 nb_etapes = int(2e5)
247 nom_fichier = "v2_essai_env_patte_fin__nb_timesteps_{}.model".format(nb_etapes)
248 
249 """
250 # Create log dir
251 log_dir = "tmp/"
252 os.makedirs(log_dir, exist_ok=True)
253 
254 env = Patte_env(False)
255 env = Monitor(env, log_dir)
256 
257 model = PPO2('MlpPolicy', env)
258 #model = PPO2('MlpPolicy', env)
```

```
#model = A2C('MlpPolicy', env)
#model = TRPO('MlpPolicy', env)

callback = SaveOnBestTrainingRewardCallback(check_freq=100, log_dir=log_dir)

model.learn(total_timesteps=nb_etapes, callback=callback)

model.save(nom_fichier)

results_plotter.plot_results([log_dir], nb_etapes, results_plotter.X_TIMESTEPS, "
    Patte")
plt.show()

del model
env.fermer()
del env
"""


env_visu = Patte_env(True)

#model_visu = PPO2.load(nom_fichier)
#model_visu = PPO2.load("tmp/best_model.zip")
#model_visu = PPO2.load("
    best_model___essai_ModifControlAX12_PPO2_Phantomx_continu___ac_CallBack____total_timesteps_2
    .0__COMPTEUR_MAX_STAGNATION_5000.zip")

#model_visu = A2C.load(nom_fichier)
#model_visu = A2C.load("tmp/best_model.zip")

model_visu = PPO2.load("tmp/best_model.zip")


nb_iterations = 1
obs = env_visu.reset()
for _ in range(nb_iterations):
    action, _states = model_visu.predict(obs)
    obs, reward, done, info = env_visu.step(action)
    print(reward)
    if done: break

del model_visu
env_visu.fermer()
del env_visu
```

Environnement Phantomx espaces continus

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-



import os

import numpy as np

import matplotlib.pyplot as plt

import gym
from gym import spaces
from gym.utils import seeding

from stable_baselines.common.policies import MlpPolicy
from stable_baselines.common.policies import CnnPolicy # réseaux convolutionnels
from stable_baselines.common.vec_env import DummyVecEnv
from stable_baselines import PPO1
from stable_baselines import PPO2
from stable_baselines import A2C
#from stable_baselines import TRPO
from stable_baselines.common.env_checker import check_env

# pour faire un suivi de l'apprentissage
from stable_baselines import results_plotter
from stable_baselines.bench import Monitor
from stable_baselines.results_plotter import load_results, ts2xy
#from stable_baselines.common.noise import AdaptiveParamNoiseSpec
from stable_baselines.common.callbacks import BaseCallback

# pré-entrainement
from stable_baselines.gail import generate_expert_traj

# modules de simulation physique
import pybullet as p
import pybullet_data

import time

from math import pi



class Phantomx_env(gym.Env):
    metadata = {
        'render.modes': ['human', 'rgb_array'],
        'video.frames_per_second': 50
    }



    def _seed(self, seed=None):
        self.np_random, seed = seeding.np_random(seed)
        return [seed]
```

```
56
57      def __init__(self, render=True):
58
59          super(Phantomx_env, self).__init__()
60
61          self.render = render
62
63          if (render):
64              self.physicsClient = p.connect(p.GUI)
65          else:
66              self.physicsClient = p.connect(p.DIRECT)  # non-graphical version
67
68          p.setAdditionalSearchPath(pybullet_data.getDataPath())  # used by loadURDF
69
70          # position initiale de la camera
71          p.resetDebugVisualizerCamera(cameraDistance=0.8, cameraYaw=0, cameraPitch
    =-30, cameraTargetPosition=[0, 0, 0])
72
73          self._seed() # initialisation de la graine (pour l'aléatoire)
74
75          p.resetSimulation()
76          p.setGravity(0, 0, -9.81)  # m/s^2
77
78          self.dt = 0.01 # pas de temps de la simulation (en secondes)
79          p.setTimeStep(self.dt)   # sec
80
81          self.periode_actions = int(5/self.dt)
82
83          self.longueur_episode = 100 # nombre d'étapes dans un épisode
84          self.compteur_etapes = 0
85
86
87          self.plane = p.loadURDF("plane.urdf")
88
89
90          # position de départ de la patte
91          self.cubeStartPos = [0, 0, 0.18]
92          self.cubeStartOrientation = p.getQuaternionFromEuler([0, 0, 0])
93
94
95          # chargement du robot dans pybullet
96          self.id_robot = p.loadURDF("urdf/phantomx_deyvi2.urdf", self.cubeStartPos,
    self.cubeStartOrientation)
97
98          # identifiant des liaisons mobiles (les moteurs)
99          self.movingJoints = [1, 3, 4, 5, 7, 8, 9, 11, 12, 13, 15, 16, 17, 19, 20,
    21, 23, 24]
100
101         # espace des actions : contrôle en position : PD control commande constante
     en vitesse
102         self.action_space = spaces.Box(low=-1, high=1, shape=(18,), dtype=np.
    float32)
103
104         # espace des observations : hauteur du bout de la patte, positions des
    moteurs
105         self.observation_space = spaces.Box(low=-1, high=1, shape=(23,), dtype=np.
```

```
        float32)
106
107
108
109     def reset(self):
110
111         self.compteur_etapes = 0
112
113         for id_moteur in self.movingJoints:
114             p.setJointMotorControl(self.id_robot, id_moteur, p.POSITION_CONTROL,
        0.0)
115             p.setJointMotorControl(self.id_robot, id_moteur, p.VELOCITY_CONTROL,
        0.0)
116
117
118         # on replace le robot à sa position initiale
119         p.resetBasePositionAndOrientation(
120             self.id_robot,
121             posObj=self.cubeStartPos,
122             ornObj=self.cubeStartOrientation
123         )
124
125         return self.deter_etat()
126
127
128     def deter_etat(self):
129
130         """
131         Détermine l'état de l'agent :
132             * Hauteur du bout de la patte
133             * Position de chaque moteur
134         """
135
136         baseOri = np.array(p.getBasePositionAndOrientation(self.id_robot))
137
138
139         JointStates = p.getJointStates(self.id_robot, self.movingJoints)
140
141         obs = np.array([baseOri[0][2],   # z base
142                         baseOri[1][0],   # orientation base
143                         baseOri[1][1],
144                         baseOri[1][2],
145                         baseOri[1][3],
146                         JointStates[0][0],  # Joint angles(Pos) -> 18
147                         JointStates[1][0],
148                         JointStates[2][0],
149                         JointStates[3][0],
150                         JointStates[4][0],
151                         JointStates[5][0],
152                         JointStates[6][0],
153                         JointStates[7][0],
154                         JointStates[8][0],
155                         JointStates[9][0],
156                         JointStates[10][0],
157                         JointStates[11][0],
158                         JointStates[12][0],
```

```
                        JointStates[13][0],
                        JointStates[14][0],
                        JointStates[15][0],
                        JointStates[16][0],
                        JointStates[17][0]], dtype=np.float32)

        return obs


    def action_moteurs(self, action):

        """
        Traduction des actions en commandes moteur
        """

        for i in range(18):
            p.setJointMotorControl2(self.id_robot, self.movingJoints[i], p.
    POSITION_CONTROL, targetPosition=pi/3*action[i], targetVelocity=1.0,
    positionGain=0.03, velocityGain=1.0)


    def est_retourne(self, tolerance_radian = 0.5):

        """
        Renvoie True si le robot s'est retourné
                False sinon
        """

        baseOri = np.array(p.getBasePositionAndOrientation(self.id_robot))

        q = np.zeros(4)
        for i in range(4):
            q[i] = baseOri[1][i]

        roll = p.getEulerFromQuaternion(q)[0]

        return (pi - abs(roll) < tolerance_radian)


    def step(self, action):

        """
        L'agent (la patte) effectue une action.
        L'environnement lui renvoie son nouvel état et sa récompense.
        Le booléen "done" permet de déterminer si l'épisode est terminé ou non.
        """

        x_avant = np.array(p.getBasePositionAndOrientation(self.id_robot))[0][0]

        self.action_moteurs(action)

        for _ in range(self.periode_actions):

            p.stepSimulation()
            if self.render: time.sleep(self.dt)

```

```
213
214         x_apres = np.array(p.getBasePositionAndOrientation(self.id_robot))[0][0]
215
216         reward = x_apres - x_avant
217
218         self.compteur_etapes += 1
219
220         obs = self.deter_etat()
221
222         done = self.compteur_etapes > self.longueur_episode
223
224         if self.est_retourne():
225             reward = -10
226             done = True
227
228         return obs, reward, done, {}
229
230
231     def render(self, mode='human', close=False):
232         pass
233
234     def fermer(self):
235         p.disconnect()
236
237
238
239 ################################### CALLBACK ########################################
240
241
242
243 class SaveOnBestTrainingRewardCallback(BaseCallback):
244     """
245     Callback for saving a model (the check is done every ``check_freq`` steps)
246     based on the training reward (in practice, we recommend using ``EvalCallback``)
    .
247
248     :param check_freq: (int)
249     :param log_dir: (str) Path to the folder where the model will be saved.
250       It must contains the file created by the ``Monitor`` wrapper.
251     :param verbose: (int)
252     """
253     def __init__(self, check_freq: int, log_dir: str, verbose=1):
254         super(SaveOnBestTrainingRewardCallback, self).__init__(verbose)
255         self.check_freq = check_freq
256         self.log_dir = log_dir
257         self.save_path = os.path.join(log_dir, 'best_model')
258         self.best_mean_reward = -np.inf
259
260     def _init_callback(self) -> None:
261         # Create folder if needed
262         if self.save_path is not None:
263             os.makedirs(self.save_path, exist_ok=True)
264
265     def _on_step(self) -> bool:
266         if self.n_calls % self.check_freq == 0:
267
```

```
268             # Retrieve training reward
269             x, y = ts2xy(load_results(self.log_dir), 'timesteps')
270             if len(x) > 0:
271                 # Mean training reward over the last 100 episodes
272                 mean_reward = np.mean(y[-100:])
273                 if self.verbose > 0:
274                   print("Num timesteps: {}".format(self.num_timesteps))
275                   print("Best mean reward: {:.2f} - Last mean reward per episode:
    {:.2f}".format(self.best_mean_reward, mean_reward))
276 
277                 # New best model, you could save the agent here
278                 if mean_reward > self.best_mean_reward:
279                     self.best_mean_reward = mean_reward
280                     # Example for saving best model
281                     if self.verbose > 0:
282                       print("Saving new best model to {}".format(self.save_path))
283                     self.model.save(self.save_path)
284 
285         return True
286 
287 
288 ################################# APPRENTISSAGE  ###################################
289 
290 
291 nb_etapes = int(2e5)
292 nom_fichier = "essai_Phantomx_env_final___nb_timesteps_{}.model".format(nb_etapes)
293 
294 """
295 # Create log dir
296 log_dir = "tmp/"
297 os.makedirs(log_dir, exist_ok=True)
298 
299 env = Phantomx_env(False)
300 env = Monitor(env, log_dir)
301 
302 model = PPO2('MlpPolicy', env)
303 #model = PPO2('MlpPolicy', env)
304 #model = A2C('MlpPolicy', env)
305 #model = TRPO('MlpPolicy', env)
306 
307 callback = SaveOnBestTrainingRewardCallback(check_freq=100, log_dir=log_dir)
308 
309 model.learn(total_timesteps=nb_etapes, callback=callback)
310 
311 model.save(nom_fichier)
312 
313 results_plotter.plot_results([log_dir], nb_etapes, results_plotter.X_TIMESTEPS, "
    Patte")
314 plt.show()
315 
316 del model
317 env.fermer()
318 del env
319 """
320 
321 
```

```
env_visu = Phantomx_env(True)

#model_visu = PPO2.load(nom_fichier)
#model_visu = PPO2.load("tmp/best_model.zip")
#model_visu = PPO2.load("
    best_model___essai_ModifControlAX12_PPO2_Phantomx_continu___ac_CallBack___total_timesteps_2
    .0__COMPTEUR_MAX_STAGNATION_5000.zip")

#model_visu = A2C.load(nom_fichier)
#model_visu = A2C.load("tmp/best_model.zip")

model_visu = PPO2.load("tmp/best_model.zip")


nb_iterations = 0 #100
obs = env_visu.reset()
for _ in range(nb_iterations):
    action, _states = model_visu.predict(obs)
    obs, reward, done, info = env_visu.step(action)
    print(reward)
    if done: break


del model_visu
env_visu.fermer()
del env_visu
```

Environnement Phantomx actions discretes

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import os

import numpy as np

import matplotlib.pyplot as plt

import gym
from gym import spaces
from gym.utils import seeding

from stable_baselines.common.policies import MlpPolicy
from stable_baselines.common.policies import CnnPolicy # réseaux convolutionnels
from stable_baselines.common.vec_env import DummyVecEnv
from stable_baselines import PPO2
from stable_baselines import A2C
#from stable_baselines import TRPO
from stable_baselines.common.env_checker import check_env

# pour faire un suivi de l'apprentissage
from stable_baselines import results_plotter
from stable_baselines.bench import Monitor
from stable_baselines.results_plotter import load_results, ts2xy
#from stable_baselines.common.noise import AdaptiveParamNoiseSpec
from stable_baselines.common.callbacks import BaseCallback

# pré-entrainement
from stable_baselines.gail import generate_expert_traj

# modules de simulation physique
import pybullet as p
import pybullet_data

import time

from math import pi

from patte import Patte


# v2 : mouvement corps (méthode)
# v3 : position de la base dans la récompense
# v4 : action : deux moteurs de deux pattes différentes
# v5 : action : 6 moteurs sur 6 pattes différentes


def num_moteur_vitesseAX12(num):
    if num == 0:
        return 102
    if num >= 1:
        return 204
    else:
        return None
```

```
56
57
58  # Pour déboguer
59  DEBUG = False
60
61  # coefficients associés à chaque terme de la récompense
62  RECOMPENSE_POIDS_COUPLE = 0.01
63  RECOMPENSE_POIDS_PROGRESSION = 10000.0
64  RECOMPENSE_POIDS_CONTACTS = 0.01
65  RECOMPENSE_POIDS_ORIENTATION = 0.1
66  RECOMPENSE_POIDS_HAUTEUR = 0.1
67
68  # distance maximale pouvant être parcourue par le robot : 3m
69  DISTANCE_MAX = 3
70
71  # délai entre deux actions successives
72  DELTA_t_ACTION = 1.0
73
74  # si le robot stagne trop longtemps au même endroit
75  COMPTEUR_MAX_STAGNATION = 5000
76
77  # hauteur initiale de la base
78  HAUTEUR_BASE_INIT = 0.18
79
80
81  class Phantomx_env(gym.Env):
82      metadata = {
83          'render.modes': ['human', 'rgb_array'],
84          'video.frames_per_second': 50
85      }
86
87
88
89      def _seed(self, seed=None):
90          self.np_random, seed = seeding.np_random(seed)
91          return [seed]
92
93      def __init__(self, render=True, video=False):
94
95          super(Phantomx_env, self).__init__()
96
97          #self.id_robot = id_robot
98          #self.physicsClient
99
100         self.render = render
101         self.video = video
102
103         if (render):
104             self.physicsClient = p.connect(p.GUI)
105         else:
106             self.physicsClient = p.connect(p.DIRECT)  # non-graphical version
107
108         p.setAdditionalSearchPath(pybullet_data.getDataPath())  # used by loadURDF
109
110         # position initiale de la camera
111         p.resetDebugVisualizerCamera(cameraDistance=0.8, cameraYaw=0, cameraPitch
```

```
    =-30, cameraTargetPosition=[0, 0, 0])
112
113         self._seed() # initialisation de la graine (pour l'aléatoire)
114
115         p.resetSimulation()
116         p.setGravity(0, 0, -9.81)  # m/s^2
117         # p.setTimeStep(1./60.)    # sec
118         self.dt = 0.01 # pas de temps de la simulation (en secondes)
119         p.setTimeStep(self.dt)    # sec
120         self.plane = p.loadURDF("plane.urdf")
121
122
123         # position de départ du robot
124         self.cubeStartPos = [0, 0, HAUTEUR_BASE_INIT]
125         self.cubeStartOrientation = p.getQuaternionFromEuler([0, 0, 0])
126         #path = os.path.abspath(os.path.dirname(__file__))
127
128         # chargement du robot dans pybullet
129         self.id_robot = p.loadURDF("urdf/phantomx_deyvi2.urdf", self.cubeStartPos,
    self.cubeStartOrientation) # phantomx_deyvi
130
131
132         ############## position possibles #################
133
134 # 1 : 380-640
135 # 3 : 200-720  # de 2 en 2 ?
136 # 4 : 380-640
137
138 # 5 : 380-640
139 # 7 : 200-720
140 # 8 : 380-640
141
142 # 9 : 380-640
143 # 11 : 200-720
144 # 12 : 380-640
145
146 # 13 : 380-640
147 # 15 : 200-720
148 # 16 : 380-640
149
150 # 17 : 380-640
151 # 19 : 200-720
152 # 20 : 380-640
153
154 # 21 : 380-640
155 # 23 : 200-720
156 # 24 : 380-640
157
158         ###################################################
159
160
161         # identifiant de chaque moteur rangés par patte et dans l'ordre base->pied
    ; puis position min et max de associées à chaque moteur
162         id_moteurs_pattes = [[1, 3, 4], [5, 7, 8], [9, 11, 12], [13, 15, 16], [17,
    19, 20], [21, 23, 24]]
163         pos_min_moteurs_pattes = [[380, 200, 380], [380, 200, 380], [380, 200,
```

```
380], [380, 200, 380], [380, 200, 380], [380, 200, 380]]
164         pos_max_moteurs_pattes = [[640, 720, 640], [640, 720, 640], [640, 720,
    640], [640, 720, 640], [640, 720, 640], [640, 720, 640]]
165
166         # identifiant des liaisons mobiles (les moteurs)
167         self.movingJoints = [1, 3, 4, 5, 7, 8, 9, 11, 12, 13, 15, 16, 17, 19, 20,
    21, 23, 24]
168
169         # definition des pattes
170         self.nb_pattes = len(id_moteurs_pattes)
171
172         self.pattes = []
173         for i in range(self.nb_pattes):
174             self.pattes.append(Patte(self.id_robot, self.physicsClient,
    id_moteurs_pattes[i], pos_min_moteurs_pattes[i], pos_max_moteurs_pattes[i]))
175
176
177         # ATTENTION !
178         # on suppose que le nombre de positions possible est le même pour chaque
    moteur !!!
179         # et que un état correspond à une position pour le moteur 1 !!
180         nb_positions_possibles = pos_max_moteurs_pattes[0][0] -
    pos_min_moteurs_pattes[0][0] + 1
181
182
183         # Espace des actions
184         actions_possibles = [3, 3, 3, 3, 3, 3, nb_positions_possibles,
    nb_positions_possibles, nb_positions_possibles, nb_positions_possibles,
    nb_positions_possibles, nb_positions_possibles]
185         self.action_space = spaces.MultiDiscrete(actions_possibles)
186
187         """
188         etats_possibles = [4, 2, nb_positions_possibles]
189         self.observation_space = spaces.MultiDiscrete(etats_possibles)
190         """
191
192         # Espace des observations
193         self.observation_space = spaces.Box(low=-1, high=1, shape=(23,), dtype=np.
    float32)
194
195         # pour le deboguage
196         self.debug_compteur = 0
197
198         # pour contrôler la stagnation du robot
199         self.compteur_stagnation = 0
200
201
202     def reset(self):
203
204         # réinitialisation des compteurs
205         self.debug_compteur = 0
206         self.compteur_stagnation = 0
207
208         # moteurs à leur position initiale
209         for i in range(self.nb_pattes):
210             self.pattes[i].reset_moteurs()
```

```
211
212         # on replace le robot à sa position initiale
213         p.resetBasePositionAndOrientation(
214             self.id_robot,
215             posObj=self.cubeStartPos,
216             ornObj=self.cubeStartOrientation
217         )
218
219         # détermination de l'état initial du robot
220         observation = self.deter_etat()
221
222         # film
223         if self.video:
224             p.startStateLogging(p.STATE_LOGGING_VIDEO_MP4, "video.mp4",
    objectUniqueIds=[0])
225
226         # on retourne l'état initial
227         return observation
228
229     def deter_etat(self):
230
231         """
232         Détermine l'état du robot :
233             * Hauteur de la base
234             * Orientation de la base
235             * Position de chaque moteur
236         """
237
238         baseOri = np.array(p.getBasePositionAndOrientation(self.id_robot))
239
240
241         JointStates = p.getJointStates(self.id_robot, self.movingJoints)
242
243         obs = np.array([baseOri[0][2],   # z base
244                         baseOri[1][0],   # orientation base
245                         baseOri[1][1],
246                         baseOri[1][2],
247                         baseOri[1][3],
248                         JointStates[0][0],  # Joint angles(Pos) -> 18
249                         JointStates[1][0],
250                         JointStates[2][0],
251                         JointStates[3][0],
252                         JointStates[4][0],
253                         JointStates[5][0],
254                         JointStates[6][0],
255                         JointStates[7][0],
256                         JointStates[8][0],
257                         JointStates[9][0],
258                         JointStates[10][0],
259                         JointStates[11][0],
260                         JointStates[12][0],
261                         JointStates[13][0],
262                         JointStates[14][0],
263                         JointStates[15][0],
264                         JointStates[16][0],
265                         JointStates[17][0]], dtype=np.float32)
```

```
266
267         return obs
268
269
270     def commande_action_patte(self, action):
271
272         """
273         Transformation du vecteur d'action en commandes sur les moteurs
274         """
275
276         for num_patte in range(6):
277
278             num_moteur = int(action[num_patte])
279             if num_moteur == 1:
280                 position_desiree_AX12 = 2*int(action[6+num_patte])
281             else:
282                 position_desiree_AX12 = int(action[6+num_patte])
283             self.pattes[num_patte].commande_action([num_moteur,
    position_desiree_AX12], num_moteur_vitesseAX12(num_moteur))
284
285
286     def mouvement_patte(self, action):
287         num_patte = int(action[0])
288         return self.pattes[num_patte].mouvement_1_moteur()
289
290     # v2 : mouvement_corps
291     def mouvement_corps(self):
292
293         """
294         Tous les moteurs executent la commande qu'ils ont reçu
295         """
296
297         for patte in self.pattes:
298             patte.mouvement_tous_moteurs()
299
300
301     def norme_vitesse_ang_base(self):
302
303         """
304         On range les vitesses correspondant aux angles d'Euler dans un vecteur
305         On renvoie sa norme
306
307         -> permet de savoir si le robot est dans une position stable ou non
308         """
309
310         BaseAngVel = p.getBaseVelocity(self.id_robot)
311
312         return np.linalg.norm([BaseAngVel[1][0], BaseAngVel[1][1], BaseAngVel
    [1][2]])
313
314     """
315       def is_fallen(self):
316
317     orientation = self.minitaur.GetBaseOrientation()
318     rot_mat = self._pybullet_client.getMatrixFromQuaternion(orientation)
319     local_up = rot_mat[6:]
```

```
320     pos = self.minitaur.GetBasePosition()
321     return (np.dot(np.asarray([0, 0, 1]), np.asarray(local_up)) < 0.85 or pos[2] <
        0.13)
322
323   def _termination(self):
324     position = self.minitaur.GetBasePosition()
325     distance = math.sqrt(position[0]**2 + position[1]**2)
326     return self.is_fallen() or distance > self._distance_limit
327     """
328     """
329     def est_tombe(self):
330             orientation = self.minitaur.GetBaseOrientation()
331             rot_mat = self._pybullet_client.getMatrixFromQuaternion(orientation)
332             local_up = rot_mat[6:]
333             pos = self.minitaur.GetBasePosition()
334             return (np.dot(np.asarray([0, 0, 1]), np.asarray(local_up)) < 0.85 or
        pos[2] < 0.13)"""
335
336
337     def est_retourne(self, tolerance_radian = 0.5):
338
339         """
340         Renvoie True si le robot s'est retourné
341                 False sinon
342         """
343
344         baseOri = np.array(p.getBasePositionAndOrientation(self.id_robot))
345
346         q = np.zeros(4)
347         for i in range(4):
348             q[i] = baseOri[1][i]
349
350         roll = p.getEulerFromQuaternion(q)[0]
351
352         return (pi - abs(roll) < tolerance_radian)
353
354     def step(self, action):
355
356         """
357         L'agent (le robot) effectue une action.
358         L'environnement lui renvoie son nouvel état et sa récompense.
359         Le booléen "done" permet de déterminer si l'épisode est terminé ou non.
360         """
361
362         # Détermination de la position du robot
363
364         baseOri = np.array(p.getBasePositionAndOrientation(self.id_robot))
365
366         x_base_avant = baseOri[0][0]
367         y_base_avant = baseOri[0][1]
368
369
370         # transformation du vecteur d'action en commandes moteur
371         self.commande_action_patte(action)
372
373         # déboguage : connaître le nombre d'itérations
```

```
374         if DEBUG: self.debug_compteur += 1
375
376         # tolerance sur la rotation de la base
377         tolerance = 0.5
378
379         # calcul de puissance
380         accu_couple = 0.0
381
382         # compteur pour imposer un temps entre chaque action
383         compteur = 0
384
385
386
387         """
388         Tant que le corps du robot est en mouvement et que le délai entre
389         chaque action n'est pas dépassé, le robot ne fait rien et on mesure
390         l'énergie consommée.
391         """
392
393         entree_boucle = True
394         en_mouvement = True
395         en_rotation = True
396
397         while entree_boucle or en_mouvement or en_rotation:
398
399             if entree_boucle: entree_boucle = False
400
401             #en_mouvement = self.mouvement_patte(action)
402             en_mouvement = self.mouvement_corps()
403             en_rotation = self.norme_vitesse_ang_base() >= tolerance
404
405             # mesure puissance
406             JointStates = p.getJointStates(self.id_robot, self.movingJoints)
407             torques = np.array([np.array(joint[3]) for joint in JointStates])
408             vitesses_ang = np.array([np.array(joint[1]) for joint in JointStates])
409             accu_couple += abs(np.dot(torques, vitesses_ang))  # peut être erreur :
    faire la somme des VA de chaque puissance moteur
410
411             p.stepSimulation()
412
413             if self.render: time.sleep(self.dt)
414
415             # si le délai est dépassé on sort de la boucle
416             compteur += 1
417             if compteur > DELTA_t_ACTION/self.dt:
418                 if DEBUG:
419                     print("\n\n\nTimeout transistion")
420                     print(self.debug_compteur, entree_boucle, en_mouvement,
    en_rotation)
421                     print("x: {}".format(x_base_avant))
422                     print("\n\n\n")
423                 break
424
425
426         """
427         RECOMPENSE_POIDS_COUPLE
```

```
428         RECOMPENSE_POIDS_PROGRESSION
429         RECOMPENSE_POIDS_CONTACTS
430         RECOMPENSE_POIDS_ORIENTATION
431         RECOMPENSE_POIDS_HAUTEUR
432         """
433 
434         # détermination des points de contact
435         ContactPoints = p.getContactPoints(self.id_robot, self.plane)
436 
437         # détermination de la position de la base
438         baseOri = np.array(p.getBasePositionAndOrientation(self.id_robot))
439 
440         # nouvelles coordonnées cartésiennes de la base
441         x_base_apres = baseOri[0][0]
442         y_base_apres = baseOri[0][1]
443 
444 
445         # récompense pour un déplacement vers l'avant
446         forward_reward = (x_base_apres - x_base_avant)/self.dt
447 
448         # pénalité pour un déplacement latéral
449         penalite_ecart = (y_base_apres - y_base_avant)/self.dt
450 
451         # pénalité proportionnelle à l'énergie consommée
452         ctrl_cost = RECOMPENSE_POIDS_COUPLE*accu_couple
453 
454         # pénalité liée aux points de contact
455         contact_cost = RECOMPENSE_POIDS_CONTACTS * len(ContactPoints)
456 
457         # pénalité liée à la hauteur de la base : si trop haute ou trop basse
458         penalite_hauteur = RECOMPENSE_POIDS_HAUTEUR*abs(baseOri[0][2] -
    HAUTEUR_BASE_INIT)
459 
460         # quaternions : orientation robot
461         q = np.zeros(4)
462         for i in range(4):
463             q[i] = baseOri[1][i]
464 
465         # pénalité liée l'orientation de la base
466         penalite_orientation = RECOMPENSE_POIDS_ORIENTATION*np.linalg.norm(p.
    getEulerFromQuaternion(q))
467 
468         # calcul de la récompense
469         reward = forward_reward - penalite_ecart - ctrl_cost - penalite_hauteur -
    penalite_orientation #- contact_cost
470         # implémenter la même récompense que Coumans à la place de 'ctrl_cost'
471         #               K * abs(produit scalaire couple/vitesse_ang) : OK
472         #              ATTENTION : 'contact_cost' retiré
473         # jouer sur les paramètres
474 
475 
476         # Détermination du nouvel état de l'agent (robot)
477         obs = np.array([baseOri[0][2],   # z base
478                         baseOri[1][0],   # orientation base
479                         baseOri[1][1],
480                         baseOri[1][2],
```

```
481                             baseOri[1][3],
482                             JointStates[0][0],  # Joint angles(Pos) -> 18
483                             JointStates[1][0],
484                             JointStates[2][0],
485                             JointStates[3][0],
486                             JointStates[4][0],
487                             JointStates[5][0],
488                             JointStates[6][0],
489                             JointStates[7][0],
490                             JointStates[8][0],
491                             JointStates[9][0],
492                             JointStates[10][0],
493                             JointStates[11][0],
494                             JointStates[12][0],
495                             JointStates[13][0],
496                             JointStates[14][0],
497                             JointStates[15][0],
498                             JointStates[16][0],
499                             JointStates[17][0]], dtype=np.float32)
500
501         # si distance maximale atteinte : épisode terminé
502         done = x_base_apres > DISTANCE_MAX
503
504
505         # ATTENTION : voir si ça vaut le coup de faire ça
506         # Si le robot stagne trop longtemps, on arrête l'épisode
507         if x_base_apres < 0.5:
508             if self.compteur_stagnation > COMPTEUR_MAX_STAGNATION:
509                 done = True
510                 self.compteur_stagnation = 0
511             else:
512                 self.compteur_stagnation += 1
513
514         # ATTENTION : reward shapping
515         if self.est_retourne():
516             done = True
517             reward = -10
518
519         # ATTENTION : reward shapping
520         if reward < -10: reward = -10
521
522         if DEBUG:
523             print("\n\n")
524             print("debug_compteur: {}".format(self.debug_compteur))
525             print("forward_reward: {}".format(forward_reward))
526             print("ctrl_cost: {}".format(ctrl_cost))
527             print("contact_cost: {}".format(contact_cost))
528             print("reward: {}".format(reward))
529             print("\n\n")
530
531         return obs, reward, done, {}
532
533
534     def render(self, mode='human', close=False):
535         pass
536
```

```
537     def fermer(self):
538 
539         if self.video:
540             p.stopStateLogging(0)
541 
542         p.disconnect()
543 
544 
545 
546 
547 ################################## CALLBACK ######################################
548 
549 
550 
551 class SaveOnBestTrainingRewardCallback(BaseCallback):
552     """
553     Callback for saving a model (the check is done every ``check_freq`` steps)
554     based on the training reward (in practice, we recommend using ``EvalCallback``)
        .
555 
556     :param check_freq: (int)
557     :param log_dir: (str) Path to the folder where the model will be saved.
558       It must contains the file created by the ``Monitor`` wrapper.
559     :param verbose: (int)
560     """
561     def __init__(self, check_freq: int, log_dir: str, verbose=1):
562         super(SaveOnBestTrainingRewardCallback, self).__init__(verbose)
563         self.check_freq = check_freq
564         self.log_dir = log_dir
565         self.save_path = os.path.join(log_dir, 'best_model')
566         self.best_mean_reward = -np.inf
567 
568     def _init_callback(self) -> None:
569         # Create folder if needed
570         if self.save_path is not None:
571             os.makedirs(self.save_path, exist_ok=True)
572 
573     def _on_step(self) -> bool:
574         if self.n_calls % self.check_freq == 0:
575 
576           # Retrieve training reward
577           x, y = ts2xy(load_results(self.log_dir), 'timesteps')
578           if len(x) > 0:
579               # Mean training reward over the last 100 episodes
580               mean_reward = np.mean(y[-100:])
581               if self.verbose > 0:
582                 print("Num timesteps: {}".format(self.num_timesteps))
583                 print("Best mean reward: {:.2f} - Last mean reward per episode:
        {:.2f}".format(self.best_mean_reward, mean_reward))
584 
585               # New best model, you could save the agent here
586               if mean_reward > self.best_mean_reward:
587                   self.best_mean_reward = mean_reward
588                   # Example for saving best model
589                   if self.verbose > 0:
590                     print("Saving new best model to {}".format(self.save_path))
```

```
591                     self.model.save(self.save_path)
592
593             return True
594
595
596
597 ################################# APPRENTISSAGE  ###################################
598
599
600 # ATTENTION : Changer le vecteur d'état !!!
601         # -> inclure les positions des pattes
602         # -> inclure l'altitude
603
604 nb_etapes = int(4e5)
605 nom_fichier = "essai_ModifControlAX12_PPO2_Phantomx_Deyvi2_mu06__v5___ac_CallBack____total_timesteps_{0}__DELTA_t_ACTION_{1}__COMPTEUR_MAX_STAGNATION_{2}.model".format(nb_etapes, DELTA_t_ACTION, COMPTEUR_MAX_STAGNATION)
606 #nom_fichier = "essai_ModifControlAX12_A2C_Phantomx_v5___ac_CallBack____total_timesteps_{0}__DELTA_t_ACTION_{1}__COMPTEUR_MAX_STAGNATION_{2}.model".format(nb_etapes, DELTA_t_ACTION, COMPTEUR_MAX_STAGNATION)
607 #nom_fichier = "essai_ModifControlAX12_TRPO_Phantomx_v5___ac_CallBack____total_timesteps_{0}__DELTA_t_ACTION_{1}__COMPTEUR_MAX_STAGNATION_{2}.model".format(nb_etapes, DELTA_t_ACTION, COMPTEUR_MAX_STAGNATION)
608
609
610 """
611 # Create log dir
612 log_dir = "tmp/"
613 os.makedirs(log_dir, exist_ok=True)
614
615 env = Phantomx_env(False)
616 env = Monitor(env, log_dir)
617
618 model = PPO2('MlpPolicy', env)
619 #model = A2C('MlpPolicy', env)
620 #model = TRPO('MlpPolicy', env)
621
622 callback = SaveOnBestTrainingRewardCallback(check_freq=1000, log_dir=log_dir)
623
624 model.learn(total_timesteps=nb_etapes, callback=callback)
625
626 model.save(nom_fichier)
627
628 results_plotter.plot_results([log_dir], nb_etapes, results_plotter.X_TIMESTEPS, "Phantomx")
629 plt.show()
630
631 del model
632 env.fermer()
633 del env
634 """
635
636
```

```
env_visu = Phantomx_env(True)

#model_visu = PPO2.load(nom_fichier)
model_visu = PPO2.load("tmp/best_model.zip")
#model_visu = PPO2.load("
    best_model___essai_ModifControlAX12_PPO2_Phantomx_v5___ac_CallBack____total_timesteps_200000
    .0__COMPTEUR_MAX_STAGNATION_5000.zip")

#model_visu = A2C.load(nom_fichier)
#model_visu = A2C.load("tmp/best_model.zip")

#model_visu = TRPO.load(nom_fichier)
#model_visu = TRPO.load("tmp/best_model.zip")

nb_iterations = 10000
obs = env_visu.reset()
for _ in range(nb_iterations):
    action, _states = model_visu.predict(obs)
    obs, reward, done, info = env_visu.step(action)
    print(reward)
    if done: break

del model_visu
env_visu.fermer()
del env_visu


"""
# pré-entrainement
# ATTENTION : pas compatible ac action Multidiscrete

env = Phantomx_env(False)

model = PPO2.load("tmp/best_model.zip")

generate_expert_traj(model, 'expert_phantomx', n_timesteps=int(1e6), n_episodes=20,
    env=env)
"""


"""
# pré-entrainement (caca)

log_dir = "tmp_pretrain_caca/"
os.makedirs(log_dir, exist_ok=True)

env = Phantomx_env(False)
env = Monitor(env, log_dir)

model = PPO2.load("tmp/best_model.zip", env)
#model.withset_env(env)

callback = SaveOnBestTrainingRewardCallback(check_freq=1000, log_dir=log_dir)

model.learn(total_timesteps=20000, callback=callback)

```

```
del model
env.fermer()
del env
"""

"""
CHIFFRES = {'0','1','2','3','4','5','6','7','8','9'}

def extraire_action(fichier):
    ligne = fichier.readline()

    act = []
    i = 0

    for _ in range(3):
        while not(ligne[i] in CHIFFRES):
            i += 1

        chiffre = str()
        while ligne[i] in CHIFFRES:
            chiffre += ligne[i]
            i += 1
        chiffre = np.int64(chiffre)
        act.append(chiffre)

    return np.array(act, dtype=np.int64)


fichier = open("exemple_parcours_PPO2_Phantomx___total_timesteps_1000000.parcours",
    "r")
env_visu = Phantomx_env(True)

nb_iterations = 10000
obs = env_visu.reset()
for _ in range(nb_iterations):
    action = extraire_action(fichier)
    obs, reward, done, info = env_visu.step(action)
    #time.sleep(1.0)
    #print(reward)
    if done: break

fichier.close()
"""
```

Contrôle des moteurs

AX12.cpp

```
/* mbed AX-12+ Servo Library
 *
 * Copyright (c) 2010, cstyles (http://mbed.org)
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

#include "AX12.h"
#include "mbed.h"

AX12::AX12(PinName tx, PinName rx, int ID, int baud)
        : _ax12(tx,rx) {
    _baud = baud;
    _ID = ID;
    _ax12.baud(_baud);

}

// Set the mode of the servo
//  0 = Positional (0-300 degrees)
//  1 = Rotational -1 to 1 speed
int AX12::SetMode(int mode) {

    if (mode == 1) { // set CR
        SetCWLimit(0);
        SetCCWLimit(0);
        SetCRSpeed(0.0);
    } else {
        SetCWLimit(0);
        SetCCWLimit(300);
        SetCRSpeed(0.0);
    }
    return(0);
}


```

```
53 // if flag[0] is set, were blocking
54 // if flag[1] is set, we're registering
55 // they are mutually exclusive operations
56 int AX12::SetGoal(int degrees, int flags) {
57
58     char reg_flag = 0;
59     char data[2];
60
61     // set the flag is only the register bit is set in the flag
62     if (flags == 0x2) {
63         reg_flag = 1;
64     }
65
66     // 1023 / 300 * degrees
67     short goal = (1023 * degrees) / 300;
68 #ifdef AX12_DEBUG
69     printf("SetGoal to 0x%x\n",goal);
70 #endif
71
72     data[0] = goal & 0xff; // bottom 8 bits
73     data[1] = goal >> 8;   // top 8 bits
74
75     // write the packet, return the error code
76     int rVal = write(_ID, AX12_REG_GOAL_POSITION, 2, data, reg_flag);
77
78     if (flags == 1) {
79         // block until it comes to a halt
80         while (isMoving()) {}
81     }
82     return(rVal);
83 }
84
85
86 // Set continuous rotation speed from -1 to 1
87 int AX12::SetCRSpeed(float speed) {
88
89     // bit 10     = direction, 0 = CCW, 1=CW
90     // bits 9-0   = Speed
91     char data[2];
92
93     int goal = (0x3ff * abs(speed));
94
95     // Set direction CW if we have a negative speed
96     if (speed < 0) {
97         goal |= (0x1 << 10);
98     }
99
100    data[0] = goal & 0xff; // bottom 8 bits
101    data[1] = goal >> 8;   // top 8 bits
102
103    // write the packet, return the error code
104    int rVal = write(_ID, 0x20, 2, data);
105
106    return(rVal);
107 }
108
```

```
109
110 int AX12::SetCWLimit (int degrees) {
111
112     char data[2];
113
114     // 1023 / 300 * degrees
115     short limit = (1023 * degrees) / 300;
116
117 #ifdef AX12_DEBUG
118     printf("SetCWLimit to 0x%x\n",limit);
119 #endif
120
121     data[0] = limit & 0xff; // bottom 8 bits
122     data[1] = limit >> 8;   // top 8 bits
123
124     // write the packet, return the error code
125     return (write(_ID, AX12_REG_CW_LIMIT, 2, data));
126
127 }
128
129 int AX12::SetCCWLimit (int degrees) {
130
131     char data[2];
132
133     // 1023 / 300 * degrees
134     short limit = (1023 * degrees) / 300;
135
136 #ifdef AX12_DEBUG
137     printf("SetCCWLimit to 0x%x\n",limit);
138 #endif
139
140     data[0] = limit & 0xff; // bottom 8 bits
141     data[1] = limit >> 8;   // top 8 bits
142
143     // write the packet, return the error code
144     return (write(_ID, AX12_REG_CCW_LIMIT, 2, data));
145 }
146
147
148 int AX12::SetID (int CurrentID, int NewID) {
149
150     char data[1];
151     data[0] = NewID;
152
153 #ifdef AX12_DEBUG
154     printf("Setting ID from 0x%x to 0x%x\n",CurrentID,NewID);
155 #endif
156
157     return (write(CurrentID, AX12_REG_ID, 1, data));
158
159 }
160
161
162 int AX12::SetBaud (int baud) {
163
164     char data[1];
```

```
165     data[0] = baud;
166
167 #ifdef AX12_DEBUG
168     printf("Setting Baud rate to %d\n",baud);
169 #endif
170
171     return (write(0xFE, AX12_REG_BAUD, 1, data));
172
173 }
174
175
176
177 // return 1 is the servo is still in flight
178 int AX12::isMoving(void) {
179
180     char data[1];
181     read(_ID,AX12_REG_MOVING,1,data);
182     return(data[0]);
183 }
184
185
186 void AX12::trigger(void) {
187
188     char TxBuf[16];
189     char sum = 0;
190
191 #ifdef AX12_TRIGGER_DEBUG
192     // Build the TxPacket first in RAM, then we'll send in one go
193     printf("\nTriggered\n");
194     printf("\nTrigger Packet\n  Header : 0xFF, 0xFF\n");
195 #endif
196
197     TxBuf[0] = 0xFF;
198     TxBuf[1] = 0xFF;
199
200     // ID - Broadcast
201     TxBuf[2] = 0xFE;
202     sum += TxBuf[2];
203
204 #ifdef AX12_TRIGGER_DEBUG
205     printf("  ID : %d\n",TxBuf[2]);
206 #endif
207
208     // Length
209     TxBuf[3] = 0x02;
210     sum += TxBuf[3];
211
212 #ifdef AX12_TRIGGER_DEBUG
213     printf("  Length %d\n",TxBuf[3]);
214 #endif
215
216     // Instruction - ACTION
217     TxBuf[4] = 0x04;
218     sum += TxBuf[4];
219
220 #ifdef AX12_TRIGGER_DEBUG
```

```
221     printf("  Instruction 0x%X\n",TxBuf[5]);
222 #endif
223 
224     // Checksum
225     TxBuf[5] = 0xFF - sum;
226 #ifdef AX12_TRIGGER_DEBUG
227     printf("  Checksum 0x%X\n",TxBuf[5]);
228 #endif
229 
230     // Transmit the packet in one burst with no pausing
231     for (int i = 0; i < 6 ; i++) {
232         _ax12.putc(TxBuf[i]);
233     }
234 
235     // This is a broadcast packet, so there will be no reply
236     return;
237 }
238 
239 
240 float AX12::GetPosition(void) {
241 
242 #ifdef AX12_DEBUG
243     printf("\nGetPosition(%d)",_ID);
244 #endif
245 
246     char data[2];
247 
248     int ErrorCode = read(_ID, AX12_REG_POSITION, 2, data);
249     short position = data[0] + (data[1] << 8);
250     float angle = ((float)(position * 300))/1024.0;
251     //float angle = (position * 300)/1024;
252 
253     return (angle);
254 }
255 
256 
257 float AX12::GetTemp (void) {
258 
259 #ifdef AX12_DEBUG
260     printf("\nGetTemp(%d)",_ID);
261 #endif
262 
263     char data[1];
264     int ErrorCode = read(_ID, AX12_REG_TEMP, 1, data);
265     float temp = data[0];
266     return(temp);
267 }
268 
269 
270 float AX12::GetVolts (void) {
271 
272 #ifdef AX12_DEBUG
273     printf("\nGetVolts(%d)",_ID);
274 #endif
275 
276     char data[1];
```

```
277     int ErrorCode = read(_ID, AX12_REG_VOLTS, 1, data);
278     float volts = data[0]/10.0;
279     return(volts);
280 }
281 
282 
283 int AX12::read(int ID, int start, int bytes, char* data) {
284 
285     char PacketLength = 0x4;
286     char TxBuf[16];
287     char sum = 0;
288     char Status[16];
289 
290     Status[4] = 0xFE; // return code
291 
292 #ifdef AX12_READ_DEBUG
293     printf("\nread(%d,0x%x,%d,data)\n",ID,start,bytes);
294 #endif
295 
296     // Build the TxPacket first in RAM, then we'll send in one go
297 #ifdef AX12_READ_DEBUG
298     printf("\nInstruction Packet\n  Header : 0xFF, 0xFF\n");
299 #endif
300 
301     TxBuf[0] = 0xff;
302     TxBuf[1] = 0xff;
303 
304     // ID
305     TxBuf[2] = ID;
306     sum += TxBuf[2];
307 
308 #ifdef AX12_READ_DEBUG
309     printf("  ID : %d\n",TxBuf[2]);
310 #endif
311 
312     // Packet Length
313     TxBuf[3] = PacketLength;     // Length = 4 ; 2 + 1 (start) = 1 (bytes)
314     sum += TxBuf[3];             // Accululate the packet sum
315 
316 #ifdef AX12_READ_DEBUG
317     printf("  Length : 0x%x\n",TxBuf[3]);
318 #endif
319 
320     // Instruction - Read
321     TxBuf[4] = 0x2;
322     sum += TxBuf[4];
323 
324 #ifdef AX12_READ_DEBUG
325     printf("  Instruction : 0x%x\n",TxBuf[4]);
326 #endif
327 
328     // Start Address
329     TxBuf[5] = start;
330     sum += TxBuf[5];
331 
332 #ifdef AX12_READ_DEBUG
```

```
333     printf("  Start Address : 0x%x\n",TxBuf[5]);
334 #endif
335 
336     // Bytes to read
337     TxBuf[6] = bytes;
338     sum += TxBuf[6];
339 
340 #ifdef AX12_READ_DEBUG
341     printf("  No bytes : 0x%x\n",TxBuf[6]);
342 #endif
343 
344     // Checksum
345     TxBuf[7] = 0xFF - sum;
346 #ifdef AX12_READ_DEBUG
347     printf("  Checksum : 0x%x\n",TxBuf[7]);
348 #endif
349 
350     // Transmit the packet in one burst with no pausing
351     for (int i = 0; i<8 ; i++) {
352         _ax12.putc(TxBuf[i]);
353     }
354 
355     // Wait for the bytes to be transmitted
356     wait (0.00002);
357 
358     // Skip if the read was to the broadcast address
359     if (_ID != 0xFE) {
360 
361 
362 
363         // response packet is always 6 + bytes
364         // 0xFF, 0xFF, ID, Length Error, Param(s) Checksum
365         // timeout is a little more than the time to transmit
366         // the packet back, i.e. (6+bytes)*10 bit periods
367 
368         int timeout = 0;
369         int plen = 0;
370         while ((timeout < ((6+bytes)*10)) && (plen<(6+bytes))) {
371 
372             if (_ax12.readable()) {
373                 Status[plen] = _ax12.getc();
374                 plen++;
375                 timeout = 0;
376             }
377 
378             // wait for the bit period
379             wait (1.0/_baud);
380             timeout++;
381         }
382 
383         if (timeout == ((6+bytes)*10) ) {
384             return(-1);
385         }
386 
387         // Copy the data from Status into data for return
388         for (int i=0; i < Status[3]-2 ; i++) {
```

```
389             data[i] = Status[5+i];
390         }
391 
392 #ifdef AX12_READ_DEBUG
393         printf("\nStatus Packet\n");
394         printf("  Header : 0x%x\n",Status[0]);
395         printf("  Header : 0x%x\n",Status[1]);
396         printf("  ID : 0x%x\n",Status[2]);
397         printf("  Length : 0x%x\n",Status[3]);
398         printf("  Error Code : 0x%x\n",Status[4]);
399 
400         for (int i=0; i < Status[3]-2 ; i++) {
401             printf("  Data : 0x%x\n",Status[5+i]);
402         }
403 
404         printf("  Checksum : 0x%x\n",Status[5+(Status[3]-2)]);
405 #endif
406 
407     } // if (ID!=0xFE)
408 
409     return(Status[4]);
410 }
411 
412 
413 int AX12::write(int ID, int start, int bytes, char* data, int flag) {
414 // 0xff, 0xff, ID, Length, Intruction(write), Address, Param(s), Checksum
415 
416     char TxBuf[16];
417     char sum = 0;
418     char Status[6];
419 
420 #ifdef AX12_WRITE_DEBUG
421     printf("\nwrite(%d,0x%x,%d,data,%d)\n",ID,start,bytes,flag);
422 #endif
423 
424     // Build the TxPacket first in RAM, then we'll send in one go
425 #ifdef AX12_WRITE_DEBUG
426     printf("\nInstruction Packet\n  Header : 0xFF, 0xFF\n");
427 #endif
428 
429     TxBuf[0] = 0xff;
430     TxBuf[1] = 0xff;
431 
432     // ID
433     TxBuf[2] = ID;
434     sum += TxBuf[2];
435 
436 #ifdef AX12_WRITE_DEBUG
437     printf("  ID : %d\n",TxBuf[2]);
438 #endif
439 
440     // packet Length
441     TxBuf[3] = 3+bytes;
442     sum += TxBuf[3];
443 
444 #ifdef AX12_WRITE_DEBUG
```

```
445     printf("  Length : %d\n",TxBuf[3]);
446 #endif
447 
448     // Instruction
449     if (flag == 1) {
450         TxBuf[4]=0x04;
451         sum += TxBuf[4];
452     } else {
453         TxBuf[4]=0x03;
454         sum += TxBuf[4];
455     }
456 
457 #ifdef AX12_WRITE_DEBUG
458     printf("  Instruction : 0x%x\n",TxBuf[4]);
459 #endif
460 
461     // Start Address
462     TxBuf[5] = start;
463     sum += TxBuf[5];
464 
465 #ifdef AX12_WRITE_DEBUG
466     printf("  Start : 0x%x\n",TxBuf[5]);
467 #endif
468 
469     // data
470     for (char i=0; i<bytes ; i++) {
471         TxBuf[6+i] = data[i];
472         sum += TxBuf[6+i];
473 
474 #ifdef AX12_WRITE_DEBUG
475         printf("  Data : 0x%x\n",TxBuf[6+i]);
476 #endif
477 
478     }
479 
480     // checksum
481     TxBuf[6+bytes] = 0xFF - sum;
482 
483 #ifdef AX12_WRITE_DEBUG
484     printf("  Checksum : 0x%x\n",TxBuf[6+bytes]);
485 #endif
486 
487     // Transmit the packet in one burst with no pausing
488     for (int i = 0; i < (7 + bytes) ; i++) {
489         _ax12.putc(TxBuf[i]);
490     }
491 
492     // Wait for data to transmit
493     wait (0.00002);
494 
495     // make sure we have a valid return
496     Status[4]=0x00;
497 
498     // we'll only get a reply if it was not broadcast
499     if (_ID!=0xFE) {
500 
```

```
501
502         // response packet is always 6 bytes
503         // 0xFF, 0xFF, ID, Length Error, Param(s) Checksum
504         // timeout is a little more than the time to transmit
505         // the packet back, i.e. 60 bit periods, round up to 100
506         int timeout = 0;
507         int plen = 0;
508         while ((timeout < 100) && (plen<6)) {
509
510             if (_ax12.readable()) {
511                 Status[plen] = _ax12.getc();
512                 plen++;
513                 timeout = 0;
514             }
515
516             // wait for the bit period
517             wait (1.0/_baud);
518             timeout++;
519         }
520
521
522         // Build the TxPacket first in RAM, then we'll send in one go
523 #ifdef AX12_WRITE_DEBUG
524         printf("\nStatus Packet\n  Header : 0x%X, 0x%X\n",Status[0],Status[1]);
525         printf("  ID : %d\n",Status[2]);
526         printf("  Length : %d\n",Status[3]);
527         printf("  Error : 0x%x\n",Status[4]);
528         printf("  Checksum : 0x%x\n",Status[5]);
529 #endif
530
531
532     }
533
534     return(Status[4]); // return error code
535 }
536
537 int AX12::instruction_simple(char instruction)
538 {
539     char TxBuf[6];
540     char sum = 0;
541     char Status[6];
542
543     TxBuf[0] = 0xff;
544     TxBuf[1] = 0xff;
545
546     // ID
547     if (instruction != ACTION)
548         TxBuf[2] = _ID;
549     else
550         TxBuf[2] = 0xfe;
551
552     sum += TxBuf[2];
553
554     TxBuf[3] = 2; // length
555     sum += TxBuf[3];
556
```

```
557     TxBuf[4] = instruction; // instruction
558     sum += TxBuf[4];
559 
560     TxBuf[5] = 0xFF - sum;
561 
562     // Transmit the packet in one burst with no pausing
563     for (int i = 0; i < 6 ; i++) {
564         _ax12.putc(TxBuf[i]);
565     }
566 
567     // Wait for data to transmit
568     wait (0.00002);
569 
570     // make sure we have a valid return
571     Status[4]=0x00;
572 
573     // we'll only get a reply if it was not broadcast
574     if (_ID!=0xFE && instruction != ACTION) {
575 
576 
577         // response packet is always 6 bytes
578         // 0xFF, 0xFF, ID, Length Error, Param(s) Checksum
579         // timeout is a little more than the time to transmit
580         // the packet back, i.e. 60 bit periods, round up to 100
581         int timeout = 0;
582         int plen = 0;
583         while ((timeout < 100) && (plen<6)) {
584 
585             if (_ax12.readable()) {
586                 Status[plen] = _ax12.getc();
587                 plen++;
588                 timeout = 0;
589             }
590 
591             // wait for the bit period
592             wait (1.0/_baud);
593             timeout++;
594         }
595 
596     }
597 
598     return(Status[4]); // return error code
599 }
600 
601 
602 
603 int AX12::pos_vit(int degrees, float vitesse, int flag)
604 {
605     char data[4];
606 
607 
608     // Position
609 
610     short goal_pos = (1023 * degrees) / 300;
611 
612     data[0] = goal_pos & 0xff; // bottom 8 bits
```

```
613     data[1] = goal_pos >> 8;    // top 8 bits
614
615
616     //Vitesse
617
618     int goal_speed = (0x3ff * abs(vitesse));
619
620     // Set direction CW if we have a negative speed
621     if (vitesse < 0) {
622         goal_speed |= (0x1 << 10);
623     }
624
625     data[2] = goal_speed & 0xff; // bottom 8 bits
626     data[3] = goal_speed >> 8;   // top 8 bits
627
628     // write the packet, return the error code
629     int rVal = write(_ID, AX12_REG_GOAL_POSITION, 4, data, flag);
630
631     return(rVal);
632 }
633
634
635 // A enlever ces commentaires :
636
637 /*
638 int AX12::SetGoal(int degrees, int flags) {
639
640     char reg_flag = 0;
641     char data[2];
642
643     // set the flag is only the register bit is set in the flag
644     if (flags == 0x2) {
645         reg_flag = 1;
646     }
647
648     // 1023 / 300 * degrees
649     short goal = (1023 * degrees) / 300;
650 #ifdef AX12_DEBUG
651     printf("SetGoal to 0x%x\n",goal);
652 #endif
653
654     data[0] = goal & 0xff; // bottom 8 bits
655     data[1] = goal >> 8;   // top 8 bits
656
657     // write the packet, return the error code
658     int rVal = write(_ID, AX12_REG_GOAL_POSITION, 2, data, reg_flag);
659
660     if (flags == 1) {
661         // block until it comes to a halt
662         while (isMoving()) {}
663     }
664     return(rVal);
665 }
666 */
667 /*
668 int AX12::SetCRSpeed(float speed) {
```

```
669
670     // bit 10      = direction , 0 = CCW, 1=CW
671     // bits 9-0    = Speed
672     char data[2];
673
674     int goal = (0x3ff * abs(speed));
675
676     // Set direction CW if we have a negative speed
677     if (speed < 0) {
678         goal |= (0x1 << 10);
679     }
680
681     data[0] = goal & 0xff; // bottom 8 bits
682     data[1] = goal >> 8;   // top 8 bits
683
684     // write the packet, return the error code
685     int rVal = write(_ID, 0x20, 2, data);
686
687     return(rVal);
688 }
689 */
690
691 //envoi d'un caractère sur le bus
692 void AX12::bus_putc(char caract)
693 {
694     _ax12.putc(caract);
695 }
```

AX12.h

```
/* mbed AX-12+ Servo Library
 *
 * Copyright (c) 2010, cstyles (http://mbed.org)
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

#ifndef MBED_AX12_H
#define MBED_AX12_H

#include "mbed.h"

//#define AX12_WRITE_DEBUG 0
//#define AX12_READ_DEBUG 0
//#define AX12_TRIGGER_DEBUG 0
//#define AX12_DEBUG 0

#define AX12_REG_ID 0x3
#define AX12_REG_BAUD 0x4
#define AX12_REG_CW_LIMIT 0x06
#define AX12_REG_CCW_LIMIT 0x08
#define AX12_REG_GOAL_POSITION 0x1E
#define AX12_REG_MOVING_SPEED 0x20
#define AX12_REG_VOLTS 0x2A
#define AX12_REG_TEMP 0x2B
#define AX12_REG_MOVING 0x2E
#define AX12_REG_POSITION 0x24

#define AX12_MODE_POSITION  0
#define AX12_MODE_ROTATION  1

#define AX12_CW 1
#define AX12_CCW 0

// ajouts :

#define PING 0x01
#define READ_DATA 0x02
#define WRITE_DATA 0x03
```

```
56 #define REG_WRITE 0x04
57 #define ACTION 0x05
58 #define RESET 0x06
59 #define SYNC_WRITE 0x83
60
61 #define REG_WRITE_FLAG 0x01
62
63 /** Servo control class, based on a PwmOut
64  *
65  * Example:
66  * @code
67  * #include "mbed.h"
68  * #include "AX12.h"
69  *
70  * int main() {
71  *
72  *   AX12 myax12 (p9, p10, 1);
73  *
74  *   while (1) {
75  *       myax12.SetGoal(0);    // go to 0 degrees
76  *       wait (2.0);
77  *       myax12.SetGoal(300);  // go to 300 degrees
78  *       wait (2.0);
79  *   }
80  * }
81  * @endcode
82  */
83 class AX12 {
84
85 public:
86
87     /** Create an AX12 servo object connected to the specified serial port, with
    the specified ID
88       *
89       * @param pin tx pin
90       * @param pin rx pin
91       * @param int ID, the Bus ID of the servo 1-255
92       */
93     AX12(PinName tx, PinName rx, int ID, int baud=1000000);
94
95     /** Set the mode of the servo
96       * @param mode
97       *    0 = Positional, default
98       *    1 = Continuous rotation
99       */
100    int SetMode(int mode);
101
102    /** Set baud rate of all attached servos
103      * @param mode
104      *    0x01 = 1,000,000 bps
105      *    0x03 =   500,000 bps
106      *    0x04 =   400,000 bps
107      *    0x07 =   250,000 bps
108      *    0x09 =   200,000 bps
109      *    0x10 =   115,200 bps
110      *    0x22 =    57,600 bps
```

```
111         *    0x67 =    19,200 bps
112         *    0xCF =     9,600 bp
113         */
114        int SetBaud(int baud);
115
116
117        /** Set goal angle in integer degrees, in positional mode
118         *
119         * @param degrees 0-300
120         * @param flags, defaults to 0
121         *    flags[0] = blocking, return when goal position reached
122         *    flags[1] = register, activate with a broadcast trigger
123         *
124         */
125        int SetGoal(int degrees, int flags = 0);
126
127
128        /** Set the speed of the servo in continuous rotation mode
129         *
130         * @param speed, -1.0 to 1.0
131         *   -1.0 = full speed counter clock wise
132         *    1.0 = full speed clock wise
133         */
134        int SetCRSpeed(float speed);
135
136
137        /** Set the clockwise limit of the servo
138         *
139         * @param degrees, 0-300
140         */
141        int SetCWLimit(int degrees);
142
143        /** Set the counter-clockwise limit of the servo
144         *
145         * @param degrees, 0-300
146         */
147        int SetCCWLimit(int degrees);
148
149        // Change the ID
150
151        /** Change the ID of a servo
152         *
153         * @param CurentID 1-255
154         * @param NewID 1-255
155         *
156         * If a servo ID is not know, the broadcast address of 0 can be used for
       CurrentID.
157         * In this situation, only one servo should be connected to the bus
158         */
159        int SetID(int CurrentID, int NewID);
160
161
162        /** Poll to see if the servo is moving
163         *
164         * @returns true is the servo is moving
165         */
```

```
166     int isMoving(void);
167 
168     /** Send the broadcast "trigger" command, to activate any outstanding
        registered commands
169      */
170     void trigger(void);
171 
172     /** Read the current angle of the servo
173      *
174      * @returns float in the range 0.0-300.0
175      */
176     float GetPosition();
177 
178     /** Read the temperature of the servo
179      *
180      * @returns float temperature
181      */
182     float GetTemp(void);
183 
184     /** Read the supply voltage of the servo
185      *
186      * @returns float voltage
187      */
188     float GetVolts(void);
189 
190 
191 
192     int read(int ID, int start, int length, char* data);
193     int write(int ID, int start, int length, char* data, int flag=0);
194 
195 
196     // Ajouts :
197 
198     int pos_vit(int degrees, float vitesse, int flag=0);
199     int instruction_simple(char instruction);
200     void bus_putc(char caract);
201 
202 private :
203 
204     //SerialHalfDuplex _ax12;
205     Serial _ax12;
206     int _ID;
207     int _baud;
208 
209 
210 };
211 
212 #endif
```

hexapode.cpp

```
1
2
3  #include "hexapode.h"
4
5  //using namespace std;
6
7  // Classe Patte
8
9
10 //Constructeur
11
12 Patte::Patte(PinName bus_tx, PinName bus_rx, int *identifiants, uint8_t nb)
13 {
14     nb_servos = nb;
15
16     for (uint8_t i = 0 ; i < nb_servos ; i++)
17         ident[i] = identifiants[i];
18
19     for (uint8_t i = 0 ; i < nb_servos ; i++)
20         servos[i] = new AX12(bus_tx, bus_rx, identifiants[i]);
21 }
22
23
24
25 // Commande en position
26
27 void Patte::positionner(int *degs)
28 {
29     for (uint8_t i = 0 ; i < nb_servos ; i++)
30         servos[i]->SetGoal(degs[i]);
31 }
32
33 int Patte::servo_set_Goal(int num_servo, int degres, int flags)
34 {
35     return servos[num_servo]->SetGoal(degres, flags);
36 }
37
38
39
40 // Mesure position
41
42 void Patte::get_position(float *dest)
43 {
44     for (uint8_t i = 0 ; i < nb_servos ; i++)
45         dest[i] = servos[i]->GetPosition();
46 }
47
48 float Patte::servo_get_position(int num_servo)
49 {
50     return servos[num_servo]->GetPosition();
51 }
52
53
54
55 // Selection Mode
```

```
56
57 void Patte::set_mode(int *modes, int *dest)
58 {
59     for (uint8_t i = 0 ; i < nb_servos ; i++)
60         dest[i] = servos[i]->SetMode(modes[i]);
61 }
62
63 int Patte::servo_set_mode(int num_servo, int mode)
64 {
65     return servos[num_servo]->SetMode(mode);
66 }
67
68
69 // Commande en vitesse
70
71 void Patte::set_CRSpeed(float *speed, int *dest)
72 {
73     for (uint8_t i = 0 ; i < nb_servos ; i++)
74         dest[i] = servos[i]->SetCRSpeed(speed[i]);
75 }
76
77 int Patte::servo_set_CRSpeed(int num_servo, float speed)
78 {
79     return servos[num_servo]->SetCRSpeed(speed);
80 }
81
82
83 // Limite position sens aiguilles montre
84
85 void Patte::set_CWLimit(int *degres, int *dest)
86 {
87     for (uint8_t i = 0 ; i < nb_servos ; i++)
88         dest[i] = servos[i]->SetCWLimit(degres[i]);
89 }
90
91 int Patte::servo_set_CWLimit(int num_servo, int degres)
92 {
93     return servos[num_servo]->SetCWLimit(degres);
94 }
95
96
97 // Limite position sens trigonométrique
98
99 void Patte::set_CCWLimit(int *degres, int *dest)
100 {
101     for (uint8_t i = 0 ; i < nb_servos ; i++)
102         dest[i] = servos[i]->SetCCWLimit(degres[i]);
103 }
104
105 int Patte::servo_set_CCWLimit(int num_servo, int degres)
106 {
107     return servos[num_servo]->SetCCWLimit(degres);
108 }
109
110 //void setID : on verra
111
```

```
112
113 // En mouvement
114
115 void Patte::is_Moving(int *dest)
116 {
117     for (uint8_t i = 0 ; i < nb_servos ; i++)
118         dest[i] = servos[i]->isMoving();
119 }
120
121 int Patte::servo_is_Moving(int num_servo)
122 {
123     return servos[num_servo]->isMoving();
124 }
125
126
127 // Mesure température
128
129 void Patte::get_Temp(float *dest)
130 {
131     for (uint8_t i = 0 ; i < nb_servos ; i++)
132         dest[i] = servos[i]->GetTemp();
133 }
134
135 float Patte::servo_get_Temp(int num_servo)
136 {
137     return servos[num_servo]->GetTemp();
138 }
139
140
141
142 // Mesure tension
143
144 void Patte::get_Volts(float *dest)
145 {
146     for (uint8_t i = 0 ; i < nb_servos ; i++)
147         dest[i] = servos[i]->GetVolts();
148 }
149
150 float Patte::servo_get_Volts(int num_servo)
151 {
152     return servos[num_servo]->GetVolts();
153 }
154
155
156 // Position vitesse
157
158 int Patte::servo_pos_vit(int num_servo, int degrees, float vitesse)
159 {
160     return servos[num_servo]->pos_vit(degrees, vitesse);
161 }
162
163 void Patte::patte_pos_vit(int *degrees, float *vitesse)
164 {
165     char trame[50];
166     short goal_pos;
167     int goal_speed;
```

```
168
169     for (int serv=0; serv < nb_servos; serv++)
170     {
171         trame[serv*5] = ident[serv]; // serv*(longueur+1) => longueur=4
172
173         goal_pos = (1023 * degrees[serv]) / 300;
174         trame[serv*5+1] = goal_pos & 0xff; // bottom 8 bits
175         trame[serv*5+2] = goal_pos >> 8;   // top 8 bits
176
177         goal_speed = (0x3ff * abs(vitesse[serv]));
178
179         if (vitesse[serv] < 0) {
180             goal_speed |= (0x1 << 10);
181         }
182
183         trame[serv*5+3] = goal_speed & 0xff; // bottom 8 bits
184         trame[serv*5+4] = goal_speed >> 8;   // top 8 bits
185     }
186
187     this->sync_write(AX12_REG_GOAL_POSITION, 4, trame);
188
189     servos[0]->instruction_simple(ACTION);
190 }
191
192 void Patte::patte_pos_vit_2(int *degrees, float *vitesse)
193 {
194     for (int i = 0; i < nb_servos; i++)
195         servos[i]->pos_vit(degrees[i], vitesse[i], REG_WRITE_FLAG);
196 }
197
198 // SYNC WRITE
199
200 void Patte::sync_write(int start, int longueur, char *trame)
201 {
202     char TxBuf[50]; // 50 : attention au sous-dimensionnement
203     char sum = 0;
204
205     TxBuf[0] = 0xff;
206     TxBuf[1] = 0xff;
207
208     // ID
209     TxBuf[2] = 0xfe; // broadcast
210     sum += TxBuf[2];
211
212     // paquet longueur totale
213     TxBuf[3] = (longueur + 1)*nb_servos + 4;
214     sum += TxBuf[3];
215
216     // instruction : SYNC WRITE => 0x83
217     TxBuf[4] = SYNC_WRITE;
218     sum += TxBuf[4];
219
220     // Start Address
221     TxBuf[5] = start;
222     sum += TxBuf[5];
223
```

```
224     // Longueur de chaque paquet de donnée
225     TxBuf[6] = longueur;
226     sum += TxBuf[6];
227 
228     for (int i=0; i < (longueur+1)*nb_servos; i++)
229     {
230         TxBuf[7+i] = trame[i];
231         sum += TxBuf[7+i];
232     }
233 
234     // checksum
235     TxBuf[7+(longueur+1)*nb_servos] = 0xff - sum;
236 
237     for (int i=0; i < 7 + (longueur+1)*nb_servos + 1 ; i++)
238         servos[0]->bus_putc(TxBuf[i]);
239 
240     // Wait for data to transmit
241     wait (0.00002);
242 }
243 
244 // ACTION
245 
246 void Patte::action()
247 {
248     servos[0]->instruction_simple(ACTION);
249 }
```

hexapode.h

```
#ifndef MBED_HEXAPODE_H
#define MBED_HEXAPODE_H

//#define HEXAPODE_DEBUG


#include "mbed.h"
#include "AX12.h"

//using namespace std;

// Classe Patte


class Patte
{
    public:

        //Constructeur
        Patte(PinName bus_tx, PinName bus_rx, int *identifiants, uint8_t nb);

        // Commande en position
        void positionner(int *degs);
        int servo_set_Goal(int num_servo, int degres, int flags=0);

        // Mesure position
        void get_position(float *dest);
        float servo_get_position(int num_servo);

        // Selection Mode
        void set_mode(int *modes, int *dest);
        int servo_set_mode(int num_servo, int mode);

        // Commande en vitesse
        void set_CRSpeed(float *speed, int *dest);
        int servo_set_CRSpeed(int num_servo, float speed);

        // Limite position sens aiguilles montre
        void set_CWLimit(int *degres, int *dest);
        int servo_set_CWLimit(int num_servo, int degres);

        // Limite position sens trigonométrique
        void set_CCWLimit(int *degres, int *dest);
        int servo_set_CCWLimit(int num_servo, int degres);

        //void setID : on verra

        // En mouvement
        void is_Moving(int *dest);
        int servo_is_Moving(int num_servo);

        // Mesure température
        void get_Temp(float *dest);
        float servo_get_Temp(int num_servo);

```

```
56          // Mesure tension
57          void get_Volts(float *dest);
58          float servo_get_Volts(int num_servo);
59
60
61          // position-vitesse
62
63          int servo_pos_vit(int num_servo, int degrees, float vitesse);
64
65
66          // commande patte en position-vitesse
67
68          void patte_pos_vit(int *degrees, float *vitesse);
69
70          void patte_pos_vit_2(int *degrees, float *vitesse);
71
72          // SYNC WRITE
73
74          void sync_write(int start, int longueur, char *data);
75
76
77          // ACTION
78
79          void action();
80
81
82
83
84      private:
85          AX12 *servos[9];
86          uint8_t nb_servos;
87          uint8_t *ident; // En cas de pépin : penser aux pbs adresses
88  };
89
90  #endif
```

Références

[1] R. O. System, ■ Tutoriel pour apprendre l'urdf : http ://wiki.ros.org/fr/urdf/tutorials ■,

[2] R. S. S. . A. G. Bartho, ■ Reinforcement learning : An introduction ■,

[3] J. K. . J. A. B. . J. Peters, ■ Reinforcement learning in robotics : A survey ■,

[4] E. C. A. I. Y. B. D. H. S. B. V. V. Jie Tan, Tingnan Zhang et E. Coumans, ■ Sim-to-real : Learning agile locomotion for quadruped robots ■,

[5] P. D. A. R. O. K. O. John Schulman, Filip Wolski, ■ Proximal policy optimization algorithms ■,

[6] P. M. M. J. P. A. John Schulman, Sergey Levine, ■ Trust region policy optimization ■,

[7] S. Levine, ■ Deep reinforcement learning (lecture) ■,

[8] R. C. F. M. A. R. Tianyu Li, Nathan Lambert, ■ Learning generalizable locomotion skills with hierarchical reinforcement learning ■,

[9] Y. ZENNIR, ■ Apprentissage par renforcement et systemes distribuÉs : Application À l'apprentissage de la marche d'un robot hexapode ■,