Camilo Ortiz – camilo.ortiz@ensta.fr
Laurent Boireau
Luc Jaulin – luc.jaulin@gmail.com

FISE25 – Robotique Autonome

Simulation, Apprentissage et Transfert au réel d'interfaces robotiques

21/08/2025

cnes
CENTRE NATIONAL D'ÉTUDES SPATIALES

ENSTA | INSTITUT POLYTECHNIQUE DE PARIS

Remerciements

Je tiens à exprimer mes sincères remerciements à toutes les personnes qui ont contribué à la réussite de mon stage.

Je suis avant tout reconnaissant envers mon tuteur Laurent BOIREAU pour son accompagnement, sa confiance et sa disponibilité tout au long de mon stage, ainsi que pour avoir partagé avec moi sa passion pour le spatial. Ses conseils m'ont permis d'approfondir mes connaissances et de découvrir de nombreux projets aussi enrichissants que passionnants.

Je tiens également à exprimer ma gratitude envers toute l'équipe du service « Systèmes Électriques et Logiciels » avec laquelle j'ai évolué, pour leur accueil chaleureux, la bonne ambiance de travail et pour m'avoir offert l'opportunité de découvrir leur métier.

J'adresse enfin mes remerciements aux différentes personnes rencontrées durant ce stage, ingénieurs comme stagiaires, pour la richesse de nos échanges, qui ont rendu cette expérience aussi agréable et motivante. Merci à Benjamin Gosselin d'avoir eu la confiance et la patience de m'apprendre à utiliser l'imprimante 3D du CNES

Table des matières

Synthèse, résumé ou abstract

L'industrie spatiale, historiquement basée sur des lanceurs à usage unique, est en pleine évolution avec le développement de lanceurs réutilisables. Ces nouveaux systèmes permettent de réduire considérablement les coûts d'accès à l'espace tout en diminuant l'impact environnemental lié à la production et à la destruction des lanceurs après chaque mission. Dans ce contexte, le CNES, l'agence spatiale française, s'est lancé dans des projets tels que Callisto et Thémis pour développer, tester et valider des technologies clés pour les futurs lanceurs réutilisables.

L'un des défis majeurs pour ces nouveaux lanceurs est l'automatisation des procédures de récupération et de sécurisation après l'atterrissage. Cela passe notamment par l'utilisation de robots, potentiellement autonomes, capables de réaliser des opérations critiques et complexes telles que les reconnexions électriques et fluidiques. Ces tâches sont essentielles pour sécuriser les lanceurs après leur retour sur Terre et préparer leur réutilisation.

Dans ce contexte, le thème de mon stage s'est articulé autour de deux objectifs principaux : l'étude des capacités d'un simulateur physique reproduisant les conditions réelles de mission et l'étude du transfert d'apprentissage des simulations vers des applications réelles.

Pour répondre à ces enjeux, le projet s'est scindé en 4 étapes clés : la simulation physique, la robotique, l'apprentissage par renforcement et le transfert vers le monde physique. Quand chacune de ces étapes est exécutée avec une certaine précision, elles forment un pipeline dont l'objectif est de fournir un système robotique robuste et fiable pouvant réaliser des missions qui demandent une grande précision de manipulation, et ce tant en simulation que dans le réel.

Nos résultats mettent en perspective la possibilité de produire de tels systèmes grâce à une technologie en pleine expansion. Il est encourageant de voir que des durées d'apprentissage relativement courtes parviennent déjà à de telles performances, et les résultats prometteurs posent les fondations d'une solution robotique autonome pour les opérations des futurs lanceurs réutilisables.

Executive Summary

The space industry, historically based on expendable launch vehicles, is undergoing a major transformation with the development of reusable launchers. These new systems significantly reduce the cost of access to space while decreasing the environmental impact associated with the production and disposal of launchers after each mission. In this context, CNES, the French space agency, has initiated projects such as Callisto and Themis to develop, test, and validate key technologies that will be needed to design future reusable launch vehicles.

One of the major challenges for these new launchers lies in automating recovery and securing procedures after landing. This involves the use of potentially autonomous robots capable of performing critical and complex operations such as electrical and fluidic reconnections. These tasks are essential to secure the launchers after their return to Earth and to prepare them for reuse.

Within this framework, my internship revolved around two primary objectives: studying the capabilities of a physical simulator reproducing real-life conditions, and investigating the transfer of learning from simulations to real-world applications.

To address these challenges, the project was divided into four key stages: physical simulation, robotics, reinforcement learning, and transfer to the physical world. When each of these stages is executed with sufficient precision, they form a pipeline whose ultimate goal is to deliver a robust and reliable robotic system capable of performing high-precision manipulation tasks both in simulation and in reality.

Our results highlight the feasibility of developing such systems through a rapidly evolving technology. It is encouraging to note that relatively short training times already yield such performance levels, and these promising results lay the groundwork for an autonomous robotic solution for operations involving future reusable launchers.

Glossaire

Charge utile

- En robotique (et plus largement en aérospatial), la charge utile désigne l'objet ou l'ensemble des équipements que le robot (ou un véhicule, comme un drone ou un satellite) est conçu pour transporter ou manipuler. Pour un bras robotique, cela correspond typiquement au poids de l'objet que le gripper peut saisir sans compromettre la stabilité ou la précision du mouvement.

Curriculum

- Le curriculum learning consiste à structurer l'apprentissage en augmentant progressivement la difficulté des tâches. Cela permet au modèle d'acquérir les compétences de base avant de traiter des tâches plus complexes, facilitant ainsi un apprentissage plus efficace et progressif.

Damping

- Le damping fait référence à la capacité d'un système à dissiper l'énergie et à réduire les oscillations ou les vibrations. En robotique, un amortissement adéquat permet d'éviter les mouvements brusques ou instables, en rendant les transitions plus fluides et stables.

End effector

- L'outil ou le composant situé à l'extrémité d'un bras robotique, qui interagit directement avec l'environnement. Cela peut être un gripper, une pince, une ventouse, un outil de soudure, etc.

Failsafe

- Mécanisme ou procédure de sécurité conçu pour ramener un système à un état sûr en cas de dysfonctionnement ou d'erreur. Dans le contexte de la robotique, cela peut signifier arrêter un robot, relâcher un objet, ou revenir à une position de sécurité.

Framework

- Structure logicielle qui fournit un ensemble de ressources, de conventions et de composants permettant de simplifier et de standardiser le développement d'applications ou de systèmes, en fournissant des fonctionnalités de base et une organisation cohérente.

Gripper

- Dispositif de préhension monté à l'extrémité d'un bras robotique, utilisé pour saisir, manipuler ou relâcher des objets.

Hyper-Paramètre

- Paramètre externe au modèle d'apprentissage (comme le taux d'apprentissage, la taille du batch, ou le nombre de couches du réseau) qui doit être défini avant l'entraînement. Ils influencent fortement les performances du modèle et sont souvent optimisés via validation croisée ou recherche automatisée.

Machine learning

- Domaine de l'intelligence artificielle qui consiste à développer des algorithmes capables d'apprendre à partir de données, sans être explicitement programmés pour chaque tâche. Le modèle ajuste ses paramètres à partir d'exemples, pour ensuite généraliser sur de nouvelles données.

Mesh

- Modèle de surface 3D représentant un objet dans un environnement de simulation ou de visualisation. Composé de sommets, d'arêtes et de faces, il définit la forme et les contours d'un objet pour la simulation physique et visuelle.

Package

- Ensemble de fichiers regroupant des scripts, bibliothèques, modèles et configurations permettant de développer une fonctionnalité ou une application dans un environnement logiciel. Dans ROS, un package peut contenir des nœuds, des configurations, des interfaces, et est organisé pour être réutilisable dans divers projets.

Plugin

- Extension logicielle qui ajoute des fonctionnalités spécifiques à une application principale, sans modifier le code source de cette application. Les plugins permettent d'ajouter des modules spécialisés et sont souvent utilisés pour augmenter les capacités d'un simulateur ou d'un logiciel de développement.

Policy (ou politique)

- En apprentissage par renforcement, une policy désigne la stratégie qu'un agent suit pour choisir ses actions en fonction de l'état de l'environnement. Elle peut être déterministe ou stochastique, et est généralement représentée par un réseau de neurones.

Sim2Real

- Abréviation de Simulation to Reality, ce terme désigne la démarche consistant à entraîner un modèle ou un robot en simulation (où les essais sont peu coûteux) puis à le transférer dans le monde réel, tout en gérant les écarts entre les deux.

Stiffness (raideur)

- En robotique ou en mécanique, la stiffness désigne la capacité d'un système ou d'un composant à résister à la déformation lorsqu'il est soumis à une force. Dans un contrôleur, ajuster la raideur permet de rendre un mouvement plus "ferme" ou plus "souple".

Acronymes

API	Application Program Interface
CALLISTO	Cooperative Action Leading to Launcher Innovation in Stage Toss-back Operations
CAO	Conception Assistée par Ordinateur
CNES	Centre National d'Etudes Spatiales
CPGE	Classes Préparatoires aux Grandes Écoles
ENSTA	École Nationale Supérieure de Techniques Avancées
GPU	Graphics Processing Unit
GUI	Graphical User Interface
IMU	Inertial Measurement Unit
JAXA	Japan Aerospace Exploration Agency
MDP	Markov Decision Process
PLAI	Policy-Level Action Integrator
PID	Proportional-Integral-Derivative
PLA	Acide Polyalactique
PPO	Proximal Policy Optimization
PROMETHEUS	Precursor Reusable Oxygen METHan cost Effective Engine
RL	Reinforcement Learning
ROS	Robot Operating System
SAPU	Simulation-Aware Policy Udpate
SBC	Sampling-Based Curriculum
SDF	Signed Distance Field
SDK	Software Development Kit
URDF	Unified Robotics Description Format
USD	Universal Scene Description

Bibliographie

[1] Nvidia, «Isaac Sim documentation,» [En ligne]. Available: https://docs.isaacsim.omniverse.nvidia.com/4.5.0/py/index.html.

[2] Nvidia, «PhysX,» [En ligne]. Available: https://docs.omniverse.nvidia.com/kit/docs/omni_usd_schema_physics/104.2/index.html.

[3] Nvidia, «Isaac Lab Documentation,» [En ligne]. Available: https://isaac-sim.github.io/IsaacLab/main/index.html.

[4] Pixar, «Universal Scene Description,» [En ligne]. Available: https://openusd.org/release/index.html.

[5] ROBOTIS, «Moteur Dynamixel XL430-250-T Documentation,» [En ligne]. Available: https://emanual.robotis.com/docs/en/dxl/x/xl430-w250/.

[6] M. L. Z. L. H. S. Xinyue Wei, «Approximate Convex Decomposition for 3D Meshes with Collision-Aware Concavity and Tree Search,» [En ligne]. Available: https://colin97.github.io/CoACD/.

[7] Trossen Robotics, «WidowX 250 S, The go-to choice for teleoperation tasks,» [En ligne]. Available: https://www.trossenrobotics.com/widowx-250.

[8] Wlkata, «Wlkata Mirobot Professional Kit - 6 Axis Robotic Arm -Ros & Matlab Simulation Teaching,» [En ligne]. Available: https://www.wlkata.com/products/professional-kit-of-wlkata-mirobot-six-axis-robot-arm-robotic-arm-k12-education-equipment.

[9] RoboDK, «Franka Emika Panda Robot,» [En ligne]. Available: Wlkata Mirobot Professional Kit - 6 Axis Robotic Arm -Ros & Matlab Simulation Teaching.

[10] Interbotix, «Open Source Robotic platforms,» [En ligne]. Available: https://www.interbotix.com/.

[11] Farama, «Gymnasium Documentation - Mujoco Ant,» [En ligne]. Available: https://gymnasium.farama.org/environments/mujoco/ant/.

[12] Nvidia, «Get started with Jetson Nano Developer Kit,» [En ligne]. Available: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit.

[13] HuggingFace, «Deep RL Course,» [En ligne]. Available: https://huggingface.co/learn/deep-rl-course/unit2/what-is-rl.

[14] A. G. B. Richard S. Sutton, Reinforcement Learning: An Introduction, 2018.

[15] C. Watkins, Learning from Delayed Rewards, 1989.

[16] Mnih, Human-level control through deep reinforcement learning, Nature, 2015.

[17] Schulman, Proximal Policy Optimization Algorithms, 2017.

[18] S. B. 3, «Stable-Baselines3 Docs - Reliable Reinforcement Learning Implementations,» [En ligne]. Available: https://stable-baselines3.readthedocs.io/en/master/.

[19] Denys88, «rl_games,» [En ligne]. Available: https://github.com/Denys88/rl_games.

[20] leggedRobotics, «rsl_rl,» [En ligne]. Available: https://github.com/leggedrobotics/rsl_rl.

[21] Tensorflow, «TensorBoard : le kit de visualisation de TensorFlow,» [En ligne]. Available: https://www.tensorflow.org/tensorboard?hl=fr.

[22] M. A. L. Bingjie Tang, «IndustReal: Transferring Contact-Rich Assembly Tasks from Simulation to Reality,» 2023.

[23] K. S. Y. Narang, «Factory: Fast Contact for Robotic Assembly,» *Robotics: Science and Sytems,* 2022.

[24] H. L. Li Cong, «Reeinforcement Learning With Vision-Proprioception Model fo rRobot Planar Pushing,» 2022. [En ligne]. Available: www.frontiersin.org/journals/neurorobotics/articles/10.3389/fnbot.2022.829437/full.

[25] S. Lee, «NumberAnalytics,» 2025. [En ligne]. Available: www.numberanalytics.com/blog/mastering-proprioception-cognitive-robotics.

1 Introduction

Je suis un étudiant en dernière année d'école d'ingénieur à l'ENSTA. Depuis mes années de CPGE je porte un intérêt pour les systèmes autonomes, l'intelligence artificielle et la programmation. C'est à l'ENSTA que j'ai trouvé la filière qui me correspondait le mieux : la voie d'approfondissement « Robotique Autonome ». J'ai suivi des cours d'automatisme, de contrôle et commande et de machine learning. Toujours poussé par la curiosité et l'envie de mieux comprendre les intelligences artificielles, j'ai orienté ma recherche de stage sur ce sujet dans l'espoir de me former et de pouvoir plus tard exercer le métier d'ingénieur en IA robotique. Je réalise mon stage de fin d'études au sein du CNES, dans le service « Systèmes Électriques et Logiciels », sur un projet innovant alliant robotique et intelligence artificielle dans le secteur du spatial.

1.1 Contexte du stage

L'industrie spatiale, depuis ses débuts, a été un domaine d'innovation et de progrès scientifiques majeurs. Elle a permis des avancées technologiques significatives, que ce soit pour l'exploration spatiale, la communication, l'observation de la Terre ou la navigation. L'espace joue un rôle central dans notre quotidien grâce à des infrastructures invisibles mais essentielles comme les satellites de communication et les systèmes de géolocalisation. Ces avancées sont le fruit d'un effort constant pour repousser les limites de nos connaissances et de nos capacités techniques, mais elles nécessitent également des moyens considérables. Le lancement de satellites et de sondes vers l'espace reste un processus extrêmement complexe et coûteux, malgré tous les progrès technologiques réalisés.

Historiquement, l'envoi de charges utiles en orbite, ou au-delà, impliquait des systèmes de lanceurs à usage unique. Ces fusées, une fois leur mission accomplie, étaient détruites dans l'atmosphère ou abandonnées dans l'espace. Bien que cette approche soit celle utilisée depuis toujours et qu'elle ait permis des missions historiques comme Apollo 11 ou les premières mises en orbite de satellites, elle est coûteuse et peu durable à long terme. En effet, chaque mission nécessite la fabrication d'un nouveau lanceur, augmentant ainsi les coûts logistiques et la complexité des lancements, tout en générant des débris spatiaux qui posent aujourd'hui des problèmes de sécurité.

Face à ces défis, une révolution dans le domaine est née : le développement de lanceurs réutilisables. L'idée de réutiliser des parties ou la totalité des fusées n'est pas nouvelle mais elle a connu des résultats concluants et prometteurs grâce à l'entreprise SpaceX. Avec la réutilisation de ses lanceurs Falcon, elle a montré que cette approche est non seulement possible mais qu'elle permet aussi de réduire les coûts d'accès à l'espace, tout en augmentant la cadence de lancement. Ce changement de conception permet d'envisager un accès plus régulier et plus abordable à l'espace.

Ainsi, dans un contexte où l'accès à l'espace devient de plus en plus stratégique, que ce soit pour des raisons économiques, géopolitiques ou scientifiques, le développement de lanceurs réutilisables apparaît comme un enjeu majeur. Il permet non seulement de rendre l'exploration spatiale plus accessible, mais aussi de soutenir le développement de nouveaux marchés comme la construction de stations en orbite basse. Le CNES, en tant qu'agence spatiale française, s'inscrit pleinement dans cette dynamique avec ses propres projets de

démonstrateurs réutilisables. Ces innovations positionnent la France et l'Europe comme des acteurs de cette nouvelle ère.

1.2 Présentation de l'entreprise

1.2.1 Historique

Le Centre National d'Études Spatiales (CNES), fondé en 1961, est l'agence spatiale française. C'est un établissement public à caractère industriel et commercial. Il combine les rôles d'agence de programme, de centre technique et d'opérateur spatial, donnant au gouvernement toutes les cartes pour définir et déployer sa stratégie spatiale. En tant que l'un des principaux contributeurs à la politique spatiale européenne, il place la France parmi les grandes puissances spatiales mondiales. En 1965, la France devient la troisième puissance spatiale mondiale, après les États-Unis et l'URSS, grâce au lancement du satellite Astérix depuis la base d'Hammaguir, en Algérie. Ce lancement propulse la France dans le cercle restreint des nations capables de concevoir, produire et envoyer des satellites dans l'espace.

Au fil des décennies, le CNES a été un moteur de l'innovation spatiale française et un acteur clé de la coopération européenne et internationale. Il a contribué à la fondation de l'Agence Spatiale Européenne (ESA) en 1975, et a joué un rôle central dans des programmes majeurs tels que Ariane, Spot, Galileo, ou encore Copernicus.

Aujourd'hui encore, le CNES demeure un acteur important du domaine du spatial au niveau mondial. Sa stratégie repose sur quatre grands objectifs. Le premier est de garantir l'autonomie stratégique de la France et de l'Europe en matière d'accès à l'espace et de développement de systèmes civils et militaires. Ensuite, il soutient l'excellence scientifique française au travers de missions spatiales internationales. Le CNES favorise également la compétitivité de l'écosystème spatial en accompagnant la transformation de la filière et en stimulant l'innovation des entreprises. Enfin, il s'engage pour un monde durable en participant à des missions spatiales visant à comprendre et à surveiller le climat, tout en encourageant la transition écologique du secteur spatial.

1.2.2 Missions et secteurs d'activité

Le CNES est chargé de proposer au Gouvernement français la politique spatiale nationale, puis d'assurer sa mise en œuvre. Ses domaines d'intervention couvrent l'ensemble du spectre de valeurs du secteur spatial.

Le CNES joue un rôle fondamental dans le développement et la supervision des lanceurs français et européens. Il est un partenaire du programme Ariane développé conjointement avec l'ESA. Il participe à la conception des lanceurs tels qu'Ariane 6 et supervise les essais, notamment au Centre Spatial Guyanais (CSG) à Kourou, le principal site de lancement européen. En parallèle, il développe la technologie du lanceur réutilisable avec les démonstrateurs Thémis et Callisto.

De plus le CNES conçoit, développe et exploite des satellites aux multiples usages tels que l'observation de la Terre, les télécommunications, la navigation et la climatologie. Il soutient activement des missions d'exploration spatiale en collaboration avec les grandes agences internationales. Il participe à des programmes scientifiques ambitieux dans les domaines de

l'astrophysique, la cosmologie ou encore la physique fondamentale et il collabore étroitement avec le Ministère des Armées pour développer et maintenir des capacités spatiales souveraines en matière de renseignement, de télécommunications sécurisées et de surveillance spatiale.

Enfin, le CNES soutient également l'émergence de nouvelles entreprises du spatial, en particulier les start-ups de la "NewSpace". Il favorise le transfert de technologie vers les filières industrielles et développe des solutions innovantes dans des domaines comme les nano-satellites, les services géospatiaux ou les systèmes embarqués.

1.2.3 Organigramme

Le CNES a une hiérarchie qui s'organise sous forme d'une grande structure en arborescence. Le Président Directeur Général est François Jacq. Les activités sont scindées en plusieurs directions. La direction dans laquelle figure mon service s'appelle la Direction Technique et Numérique (DTN). Celle-ci aussi se scinde en plusieurs sous-directions comme la Direction des Systèmes de Transport Spatial (STS) chapeautée par Christophe Bonhomme. Enfin, le service où j'ai effectué mon stage « Systèmes Electriques et Logiciels » (SEL) se concentre sur tous les aspects software embarqué des systèmes de transport. Les ingénieurs travaillent sur des domaines tels que l'avionique, les systèmes électriques, la télémesure, les services GNSS pour le positionnement...

Mon tuteur Laurent Boireau est spécialiste en architecture numérique et en simulation au sein du service SEL. Intéressé depuis longtemps par l'astrophysique et par les moyens d'accès à l'espace (les fusées), il a réalisé ses études d'ingénieur et son doctorat dans le domaine du spatial. Son premier travail au CNES a été un Post Doc en aérodynamique. Ensuite il a obtenu un emploi dans l'entreprise en partie grâce à sa formation d'ingénieur électronicien. Il a travaillé dans de multiples projets, Perseus, Themis ou encore Ariane 5.

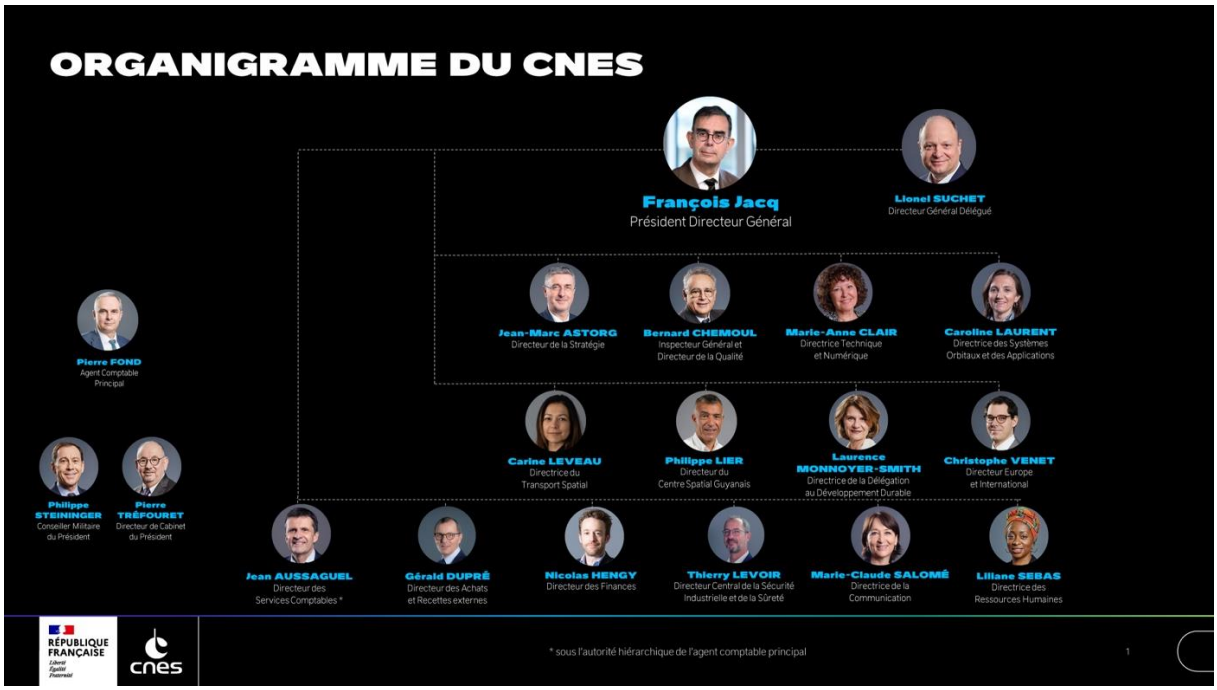


Figure 1 : Organigramme du CNES

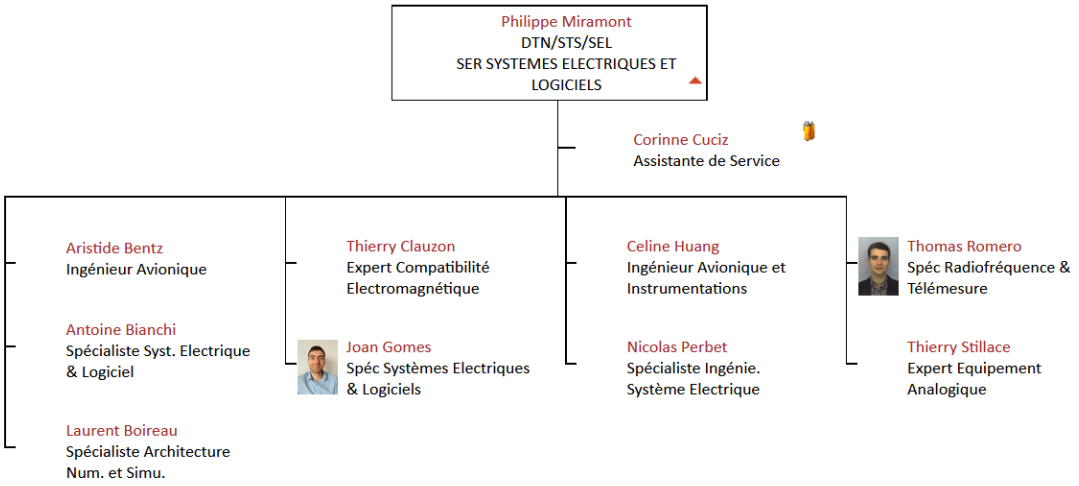


Figure 2 : Organigramme du service DTN/STS/SEL

1.3 État de l'art et analyse du besoin

1.3.1 Objectifs

Le CNES, et plus précisément la Direction des Lanceurs, a pour but de concevoir des systèmes lanceurs pour les fusées Ariane et Véga. Elle est également responsable du développement des lanceurs du futur avec notamment les démonstrateurs de lanceurs réutilisables Thémis et Callisto. Depuis le début des années 1980, le CNES travaille sur le concept de réutilisabilité, ce qui a amené à des avancées concrètes au fil des années. En 2017, après une phase d'étude interne, le CNES a décidé, en collaboration avec des partenaires internationaux, de lancer un projet visant à effectuer plusieurs vols d'essai avec un petit véhicule spatial entièrement réutilisable, nommé Callisto.

Callisto, acronyme de « Cooperative Action Leading to Launcher Innovation in Stage Toss-back Operations », est un projet de coopération entre le CNES, la JAXA, l'agence spatiale japonaise et le DLR, l'agence spatiale allemande. Ce démonstrateur a pour objectif de réaliser des tests en vol pour mieux maîtriser la récupération complexe d'un lanceur, optimiser les opérations de remise en état entre deux vols et évaluer de manière précise le coût d'un futur lanceur européen réutilisable. Cependant, Callisto n'est pas destiné à devenir un véhicule opérationnel. Mesurant 13 mètres de hauteur pour un mètre de diamètre, il est équipé d'un moteur cryotechnique réutilisable alimenté à l'hydrogène et à l'oxygène. Le premier vol d'essai de Callisto est prévu en 2025 et sera le premier d'une série de dix vols.

En parallèle, le projet européen Thémis a été initialement développé par ArianeWorks, la plateforme d'innovation CNES/ArianeGroup, pour préfigurer le premier étage du futur lanceur Ariane Next, successeur d'Ariane 6. Il est aussi prévu pour être utilisé comme le premier étage du lanceur Maïa, un projet français de lanceur spatial léger partiellement réutilisable. Une première phase de développement a permis de réaliser des essais fonctionnels de mise à feu du premier moteur PROMETHEUS. Ce démonstrateur d'étage réutilisable, de 28 mètres de hauteur et 3,5 mètres de diamètre, vise à tester des technologies clés pour les lanceurs

réutilisables telles que le système d'atterrissage, le guidage, la navigation et le contrôle, la surveillance des structures et systèmes ainsi que la sécurisation du véhicule via des robots.

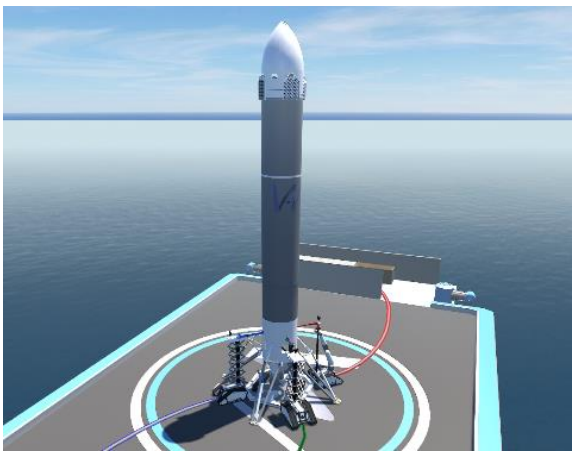


Figure 3 : Vue d'artiste de Thémis

Une des compétences clés pour garantir le succès de la réutilisation est le développement de moyens robotiques, potentiellement autonomes, capables de participer aux opérations de préparation, de récupération, d'inspection et de remise en état des lanceurs. Dans le cas de SpaceX, après chaque lancement, le premier étage du Falcon 9 retourne sur Terre en effectuant un atterrissage contrôlé, soit sur une barge en mer, soit sur une plateforme terrestre. Une fois au sol, les robots autonomes interviennent pour sécuriser l'étage, analyser son état et le préparer pour un prochain lancement. Ces robots, dotés de capteurs avancés et de systèmes de perception, sont capables de manipuler des outils et de détecter des anomalies structurelles sur le lanceur. Ils jouent ainsi un rôle crucial dans la réduction des coûts de maintenance entre les missions.

Comme pour SpaceX, les robots autonomes utilisés pour la passivation de lanceurs réutilisables seront également responsables de l'optimisation des processus de récupération. Les systèmes robotiques permettent notamment de réduire l'exposition des opérateurs à des environnements dangereux, comme les zones à haute température ou celles exposées à des résidus de carburants. Cela assure non seulement une plus grande sécurité pour les équipes au sol mais aussi l'amélioration de l'efficacité des opérations en minimisant les risques d'erreurs humaines. Ces robots auront donc pour mission de sécuriser le lanceur une fois son retour sur Terre et de le préparer au prochain vol. On compte parmi leurs tâches la vidange des réservoirs et la recharge des batteries, qui sont des missions critiques et complexes car elles demandent une haute précision dans le contrôle du robot.

1.3.2 Reformulation du problème dans son contexte

La réalisation de ce stage se concentre sur l'étude de la capacité des robots autonomes à réaliser des missions complexes telles que les reconnexions électriques et fluidiques, dans des environnements contraints et sans assistance humaine directe. Il s'agit d'évaluer les possibilités de mise en œuvre de solutions robotiques capables d'accomplir des tâches de manière fiable, en tenant compte des contraintes de l'environnement et des exigences opérationnelles spécifiques au domaine spatial.

Dans un premier temps, ce stage vise à étudier en profondeur les capacités d'un simulateur physique capable de reproduire avec précision les conditions réelles d'opérations. Ce simulateur doit inclure des modèles détaillés de forces, de friction et d'interactions mécaniques, afin de créer un environnement fidèle à la réalité. En plus de permettre le test et l'optimisation des algorithmes de contrôle et de planification, il pourra intégrer des robots potentiellement autonomes qui seront testés dans des scénarios variés. Il offrira aussi la possibilité de modifier dynamiquement les paramètres physiques environnementaux afin de simuler divers scénarios réalistes.

Le second objectif est d'évaluer les performances d'un robot entraîné en simulation *via* des techniques de machine learning et d'estimer la faisabilité du transfert de cet apprentissage vers des applications en conditions réelles. Il s'agit d'une problématique particulièrement complexe car ce type de transfert nécessite une grande précision dans la simulation d'environnements réels. Cette difficulté est d'autant plus importante dans le domaine spatial, où les tests physiques sont rares, coûteux et limités par les contraintes matérielles et logistiques.

Si aujourd'hui les robots B1 et B2 sont télé opérés et/ou automatisés à l'aide d'algorithmes de contrôle standards, ce stage cherche à fournir un aperçu de l'efficacité des méthodes d'apprentissage sur des missions aussi complexes et précises que le branchement de connecteurs en milieu hasardeux, missions que nous étudierons en priorité durant ce stage.

Figure 4 : Robot manipulateur réalisant les connections

2 Démarche

2.1 Simulation

2.1.1 Le simulateur

Le simulateur pour lequel nous avons opté est Isaac Sim [1]. Il s'agit d'une plateforme de simulation robotique développée par Nvidia, conçue pour créer, tester et valider des systèmes robotiques dans des environnements virtuels. En ayant la puissance de calcul graphique adéquate, elle permet de simuler des interactions complexes avec précision tout en ayant la possibilité d'intégrer des frameworks de robotique tels que ROS2.

Isaac Sim possède un moteur de physique très avancé, nommé PhysX [2] et développé aussi par Nvidia. C'est un moteur capable de simuler des robots dans des environnements complexes avec des interactions physiques précises. Il permet de modéliser des scénarios variés allant de la manipulation d'objets à la navigation autonome, en prenant en compte des éléments comme la friction, les collisions, les forces et la gravité.

Isaac Sim s'intègre aussi facilement avec ROS2, ce qui permet de tester les algorithmes dans un environnement virtuel avant de les déployer sur des robots physiques. Il est ainsi possible de faire le lien entre les outils de simulation et le contrôle réel des robots, en exploitant des capteurs simulés nativement sur Isaac Sim, comme les caméras et les LiDARs.

Enfin, cette plateforme est adaptée pour l'entraînement de robots via des techniques d'apprentissage par renforcement, notamment grâce à l'intégration de frameworks comme Isaac Lab [3]. Il est ainsi possible d'entraîner des modèles en simulant des millions d'itérations dans des environnements virtuels.

Tous ces attributs font d'Isaac Sim un outil puissant et complet pour concevoir, tester et valider des systèmes robotiques complexes dans des environnements simulés avec précision. Il correspond aux besoins du projet en offrant des interfaces adaptées pour l'apprentissage par renforcement de ces robots.

2.1.2 La création d'environnement

La création d'environnements dans Isaac Sim, appelés scènes, se base sur un format de fichier Universal Scene Description (USD) [4]. Une scène représente l'espace en trois dimensions dans lequel les objets et le robot interagissent.

Pour commencer, il est essentiel de configurer certains paramètres pour garantir que la simulation reflète le plus fidèlement possible les conditions réelles. Parmi ces paramètres figurent la *Physic Scene* qui gère la simulation des lois physiques, le *ground plane* représentant le sol, ainsi que les sources de lumières pour éclairer la scène. Avec l'aide de PhysX on configure la physique de la scène en ajustant des éléments comme la gravité, les forces et les frictions.

Ensuite, il est possible d'ajouter des objets à la scène grâce aux outils intégrés d'Isaac Sim. Par exemple, pour créer un simple cube bleu, il faut d'abord définir ses dimensions, sa position et

son orientation puis lui attribuer la propriété physique *Rigid Body with Colliders Preset*. Elle permet notamment de définir la masse du cube et de créer un « mesh » de collision pour simuler les interactions physiques. Il est également possible de définir la couleur et d'autres propriétés physiques en appliquant un matériau personnalisé.

Des robots peuvent aussi être ajoutés à la scène, soit en utilisant des modèles prédéfinis fournis par Isaac Sim au format USD, soit en les important *via* des fichiers URDF (Universal Robot Description File). Il est aussi possible d'intégrer des capteurs pour simuler des systèmes de perception. Ces capteurs permettent d'obtenir des informations sur l'environnement qui pourront être utilisées par des algorithmes de perception. L'un des atouts d'Isaac Sim est sa capacité à générer des simulations visuelles réalistes permettant d'utiliser des caméras et les mêmes algorithmes de perception que dans les conditions réelles.

Enfin, il est possible d'interagir avec l'environnement pour définir des actions du robot ou récupérer les informations des capteurs. Pour cela, il faut utiliser des scripts Python ou des *Action Graphs*. Ces derniers sont des graphes constitués de nœuds représentant des actions ou des états, et des connexions entre ces nœuds qui définissent l'enchaînement des événements ou des signaux. Par exemple, un nœud peut représenter la lecture d'un capteur, tandis qu'un autre représente une commande en mouvement. Il est aussi possible d'ajouter des nœuds comprenant du code Python pour des besoins spécifiques.

Pour construire ces environnements, l'interface d'Isaac Sim nous donne le choix entre utiliser le GUI pour placer manuellement les objets dans la scène et paramétrer la physique de l'ensemble, ou bien passer entièrement par le script en python. Selon Nvidia, tout ce qui est faisable sur l'interface graphique est réalisable aussi en script python alors que l'inverse n'est pas vrai. C'est pourquoi nous avons fait le choix d'apprendre à maîtriser l'API python. Cela nous a permis d'avoir une meilleure vue sur tous les paramètres physiques, et surtout d'avoir la possibilité tout au long du stage de changer très facilement des détails dans la scène. Tous nos robots ont donc été entièrement scriptés.

Une fois le robot modélisé, il est tout à fait possible de piloter ses joints, de lire les données de ses capteurs, de récupérer les positions dans le monde de chacun des membres du robot, tout cela grâce à l'API. Si l'utilisateur souhaite modifier un paramètre physique du robot (comme sa masse, sa collision…) cela est possible également. Pour cela, Isaac Sim emploie des *views*. Ce sont des classes python qui constituent une interface entre le moteur physique et l'utilisateur de l'API.

2.1.3 Étude physique préliminaire du simulateur

Par la suite, nous avons réalisé tout un travail d'étude de ce simulateur. Il s'agissait ici de bien comprendre le fonctionnement, la physique et les performances de l'outil, de le prendre en main, et surtout de fournir un travail préalable à la conception d'environnements de simulation les plus fidèles à la réalité possible.

2.1.3.1 Étude des liaisons physiques

D'abord nous avons étudié les interactions de contact et la modélisation des efforts dans Isaac Sim. Pour comprendre comment les efforts sont calculés, nous avons étudié un joint rotatif

basique et le Controller PID qui est associé. La formule donnée par Nvidia dans sa documentation est

$$force = stiffness\ .ecart_{pos} + damping\ .ecart_{vitesse}$$

Nous avons testé cela avec un premier robot, une version simplifiée du *RRbot* qui est un robot constitué de trois tiges reliées par deux joints révolus (Revolute Revolute Bot en anglais). Le nôtre était plus basique : il ne comprenait qu'une simple liaison pivot.

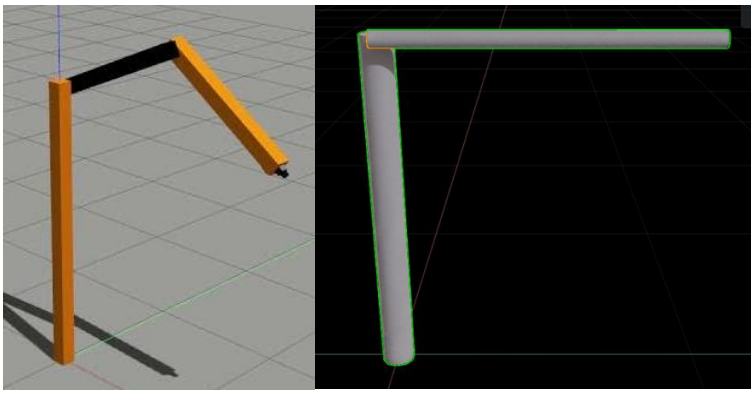


Figure 5 : RRbot (à gauche) et Rbot (à droite)

La compréhension du calcul des efforts nous a permis à la fois de comprendre le fonctionnement interne du simulateur, de voir comment utiliser l'API Python pour écrire nos propres contrôleurs PID et ainsi de concevoir avec précision un modèle simulé de servomoteur que nous possédons au laboratoire, et ce de manière très fidèle à la réalité. Cela nous sera utile car les robots réels sur lesquels nous travaillons sont équipés de ces mêmes servomoteurs. Ce sont les modèles Dynamixel XL430 et 2XL430 de chez ROBOTIS [5].

Figure 6 : Moteur Dynamixel 2XL430-250-T (Modèle simulé à gauche, modèle réel à droite)

2.1.3.2 Étude des interactions de contact

Par la suite, afin d'analyser les interactions de contact et de frottement statique et dynamique nous avons modélisé un autre robot légèrement plus complexe car muni d'une pince avec deux degrés de liberté : un joint prismatique pour descendre la pince et un joint de rotation pour ouvrir ou fermer la pince.

L'objectif de la pince est de ramasser le cube qui est à ses pieds. C'est une action assez simple qui est très facilement programmable empiriquement. Pour concevoir ce robot simple nous nous sommes inspiré des machines à jeu japonaises dans lesquelles il faut contrôler une pince similaire pour obtenir un cadeau.

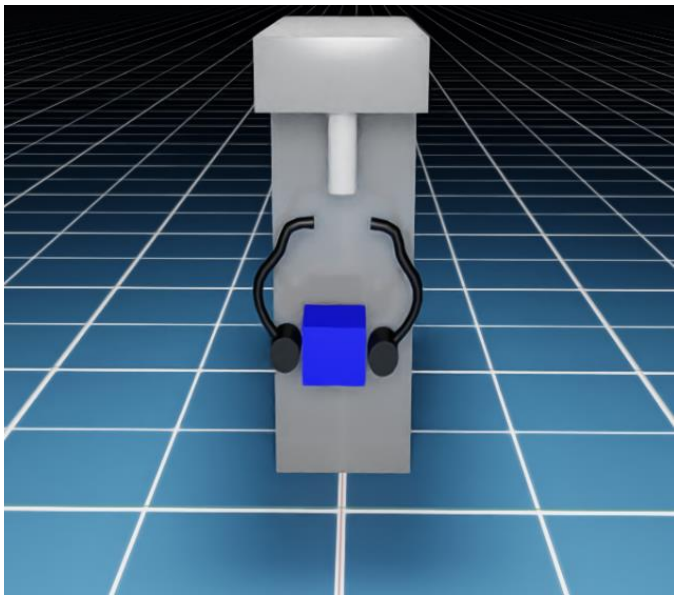


Figure 7 : Robot Pince tenant un cube

A l'aide de ce robot nous avons mené deux séries de tests pour nous aider à mieux comprendre la physique des contacts décrite dans Isaac Sim. Chaque banc teste un unique paramètre. Tous les autres sont fixés et seul le paramètre en question augmente légèrement à chaque nouvel essai du robot :

- En fixant la valeur de l'effort de préhension de la pince sur le cube, nous avons fait varier les coefficients de frottement statique et dynamique entre les doigts de la pince et le cube. L'objectif était de voir à partir de quelle valeur de frottement le robot était capable d'attraper le cube sans que celui-ci ne lui glisse entre les doigts.
- Nous avons ensuite fixé la valeur du coefficient de frottement assez bas pour que le cube se comporte comme un glaçon d'apéritif et nous avons cherché l'effort nécessaire au robot pour attraper le cube.

Effort appliqué (N.m) (friction=0,2)	Attrape le cube ?	**Coefficient de friction (Effort=15N.m)**	Attrape le cube ?
0.0 – 12	Non	0.0 – 0.15	Non
12 – 20	Presque	0.15 – 0.3	Presque
20 – 25	A peine	0.3 – 0.45	A peine
> 25	Oui	> 0.45	Oui

Figure 8 : Résultat des tests en effort (à gauche) et en friction (à droite)

Ces résultats proviennent de la simple observation du comportement du simulateur. « Non » indique que le robot glisse sur le cube et n'arrive pas du tout à le soulever. « Presque » indique que le cube est soulevé mais glisse et finit par retomber. Avec « A peine » on sait que le cube glisse légèrement entre les doigts du robot mais reste néanmoins en l'air. Et avec « Oui » on a une pince qui attrape parfaitement le cube sans le laisser glisser.

Ces bancs de tests physiques ont permis d'étudier les composantes physiques des objets qui seront ensuite manipulés par le robot et d'ajuster leurs paramètres pour s'approcher de la réalité.

2.1.3.3 Collisions

Une bonne approximation des collisions est nécessaire pour fidéliser le simulateur. Mais à l'inverse une trop bonne approximation est néfaste car trop gourmande en ressources computationnelles. Lorsqu'on crée une forme ou qu'on importe un mesh spécifique dans Isaac Sim, on peut choisir de lui donner une propriété de collision. Le moteur physique les prendra en compte et ajoutera des efforts en fonction de paramètres tels que la vitesse de collision entre deux objets, les coefficients de frottement…

Pour un ordinateur, calculer une collision entre deux faces est lourd et complexe. Évidemment, plus il y a de vertex dans l'objet, plus il y a de faces et plus le calcul de la collision est long. C'est pourquoi il est nécessaire de créer un nouveau mesh de collision, séparé du mesh visuel. Pour faire cela il y a deux solutions : la première consiste à créer un mesh primitif (un cube, un cylindre… proposés nativement par Isaac Sim) et le paramétrer pour que sa forme corresponde à peu près à l'objet auquel on veut assigner la collision. La seconde consiste à prendre le mesh visuel et à le décomposer selon des algorithmes spécifiques pour alléger la charge de calcul.

Le simulateur comprend plusieurs algorithmes d'approximation de mesh. Le plus simple et léger s'appelle *Convex Hull*. Il consiste à englober le mesh dans une boîte la plus contractée possible. Cette approximation est parfaite pour tout objet naturellement convexe, comme un cube ou une sphère, par contre elle est très mauvaise si l'objet possède des courbures. *Convex Decomposition* [6] offre un bon compromis entre performance de calcul et approximation. Elle consiste à faire une décomposition en éléments convexes du mesh. Donc plutôt que d'avoir une boîte qui englobe le mesh, on a plusieurs petites boîtes collées entre elles.

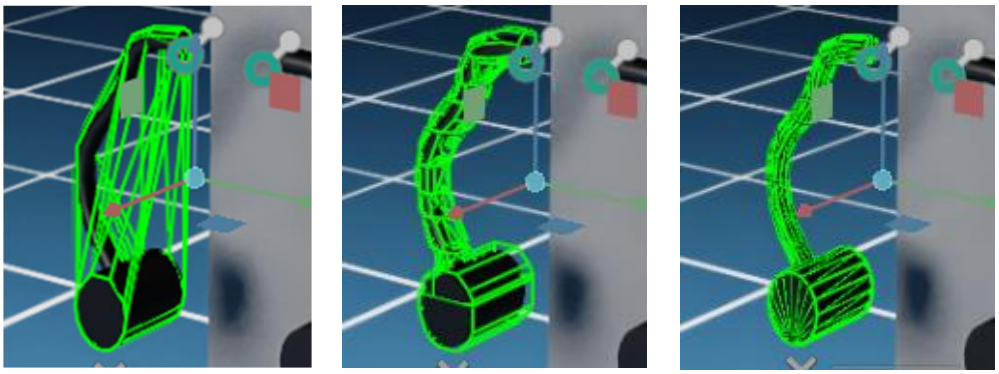


Figure 9 : Exemples d'approximations. Dans l'ordre, Convex Hull, Convex Decomposition, et SDF Decomposition

La mise en place du simulateur a permis de créer des environnements particulièrement fidèles à la réalité, tant sur le plan visuel que physique. Grâce à la précision des interactions

mécaniques modélisées, telles que les forces, frictions et collisions, il est possible de réaliser des tests approfondis en reproduisant des conditions de manipulation proches de celles du monde réel. De plus, le réalisme visuel du simulateur donne la possibilité d'évaluer des algorithmes de perception en simulant le comportement des capteurs de manière fiable.

2.2 Robotique/ Conception

2.2.1 Démarche

Une fois le simulateur prêt, il nous a fallu travailler sur l'aspect robotique du stage. Notre robot est un bras robotique à base fixe nommé Widow. Son objectif est, une fois placé sur un système mobile tel que les actuels robots B1 et B2, de réaliser des opérations de connexion complexes telles que des prises « socket électrique » ou encore « prise fluidique cryo quart de tour ». C'est notre démonstrateur de solution robotique entraîné à l'apprentissage par renforcement pour réaliser ces connexions sur le lanceur après atterrissage. Afin de nous faire la main sur cette méthode d'apprentissage et son transfert au robot réel, nous avons travaillé sur un robot « intermédiaire » en termes de complexité. C'est un robot nommé Spider, doté de 4 pattes et dont l'objectif premier est de se déplacer en ligne droite en copiant la démarche d'une araignée. L'intérêt de ce robot est qu'il est relativement facile à entraîner (c'est un cas école) et qu'il met en jeu des interactions mécaniques et de contact beaucoup plus simples que le bras robotique (interactions entre les pattes et le sol horizontal).

2.2.2 Widow

2.2.2.1 Présentation du robot

Figure 10 : Robot Widow

L'objet principal du stage est un bras robotique dont le nom complet est WidowX250s [7]. C'est un bras conçu par Trossen Robotics et fabriqué avec des moteurs Dynamixel. Il possède 6 degrés de liberté en plus de son gripper et est à base fixe. L'intérêt de ce robot est qu'il se place à mi-chemin entre un petit robot de laboratoire comme le Wlkata [8] et des gros robots industriels comme le Franka [9]. Il est donc parfait pour servir de démonstrateur de solution robotique entraînée à l'IA.

Le robot mesure environ 65 cm de haut et peut évoluer dans une demi-sphère de 130cm de diamètre. Sa charge utile peut aller jusqu'à 250g. Son intégration à des frameworks tels que ROS2, moveIt, Gazebo est très largement facilitée grâce aux travaux effectués par la plateforme de développement robotique open source Interbotix [10].

La première mission de ce robot est « pick and place », il s'agit de réussir à attraper un objet simple (un cube par exemple) et de le placer dans un endroit donné dans l'espace. La réalisation de cette mission n'est qu'un pas vers l'objectif réel du robot qui est de réaliser des connexions type prise électrique et quart de tour.

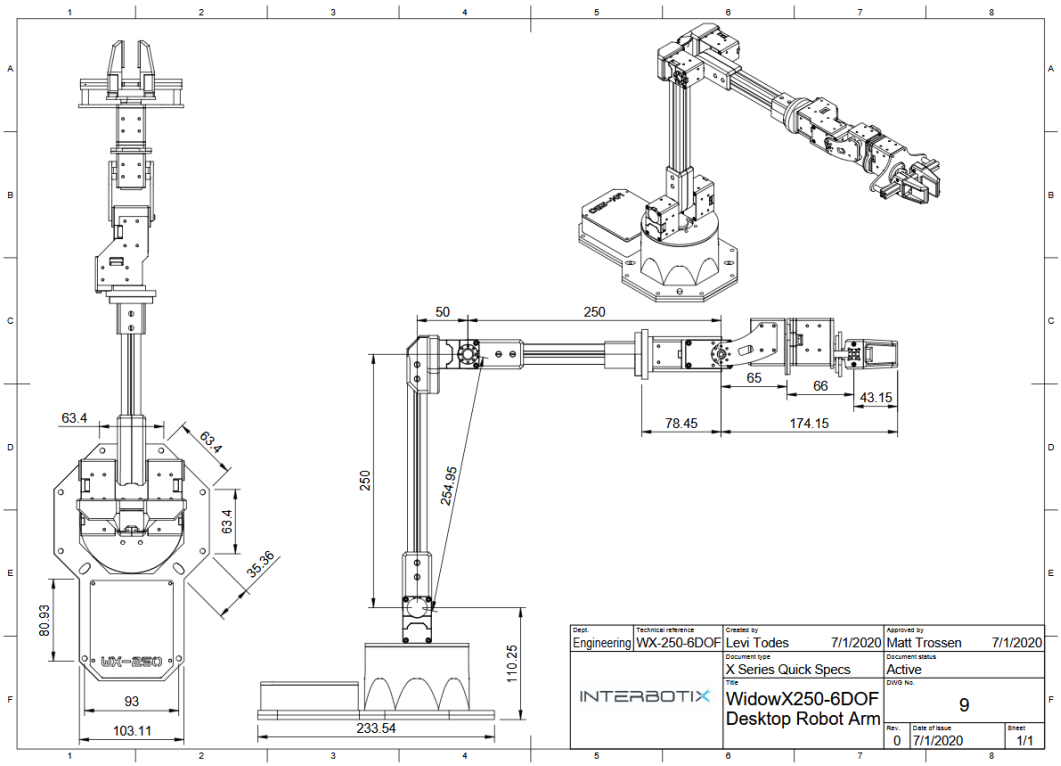


Figure 11 : Dessin technique du WidowX250

2.2.2.2 Modélisation dans Isaac Sim

Le Widow est très facile à charger dans Isaac Sim grâce à Interbotix qui fournit le fichier URDF ainsi que tous les fichiers de mesh. Isaac Sim embarque un plugin qui permet de charger les fichiers URDF et de les convertir en USD.

Cependant nous avons rencontré quelques problèmes de « *malloc()* » qui proviennent sûrement du fait que l'espace mémoire alloué à l'entraînement avec Isaac Lab est surchargé par cette modélisation du robot. De plus, bien que ne possèdant pas la pince qui est originellement utilisée par le robot, nous avons souhaité avoir le contrôle sur la pince à utiliser. C'est pourquoi nous avons finalement décidé de scripter entièrement le robot via l'API python d'Isaac Sim. Cela nous a permis aussi de retravailler les meshs de collision et de réduire les erreurs de *malloc*.

Concernant la pince, nous avons réalisé un modèle simple et adapté à la monture du gripper. En parallèle nous avons chargé ce modèle dans la simulation. Dans le modèle simulé, la pince est faite avec deux joints prismatiques qui actionnent chacun un doigt de la pince. Dans la réalité, il n'y a qu'un seul servomoteur qui sert à actionner de manière symétrique les deux doigts au moyen d'un pantographe.

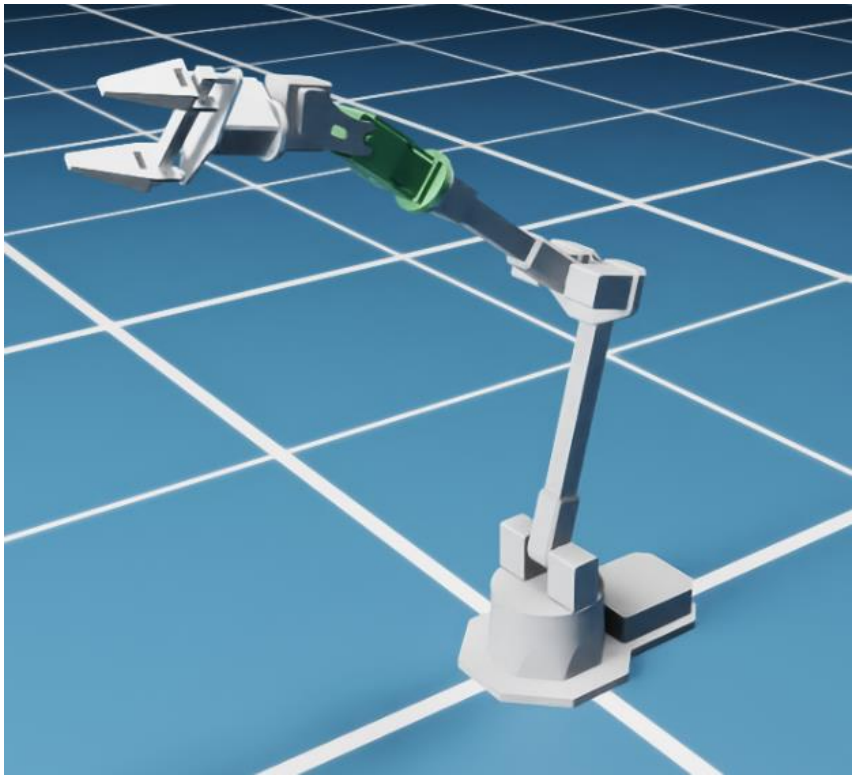


Figure 12 : Modèle simulé de Widow dans Isaac Sim

Figure 13 : Gripper de Widow avec la pince imprimée (en gris)

2.2.2.3 « Pick cube » avec cinématique inverse

Comme premier test, nous avons programmé la mission du robot avec de la cinématique inverse. Il s'agit d'une méthode mathématique permettant de déterminer pour chaque articulation du bras la position nécessaire afin que le bout du robot, l'*end effector* arrive à une position donnée.

Cette méthode est très fonctionnelle pour une mission du type « pick cube » car il suffit de programmer le robot pour qu'il aille jusqu'au cube, qu'il ferme le grip puis qu'il soulève le cube pour le déposer à un autre endroit. Mais cette méthode manque de finesse pour des tâches plus complexes telles que le branchement de prises électriques.

2.2.3 Spider

2.2.3.1 Présentation du robot

Dans l'optique de tester les méthodes d'apprentissage et de transfert au réel, nous avons voulu nous faire la main sur un prototype de robot araignée. Pour cela nous nous sommes inspirés du robot Ant de chez Mujoco [11]. Ant est un système relativement simple doté d'un corps principal (le torse) et de quatre pattes disposées sur les 4 points cardinaux du corps. Chaque patte est constituée de deux membres. Le premier membre est relié au corps par une liaison pivot sur l'axe vertical. Le second est relié au premier membre par une liaison pivot orthogonale à la première.

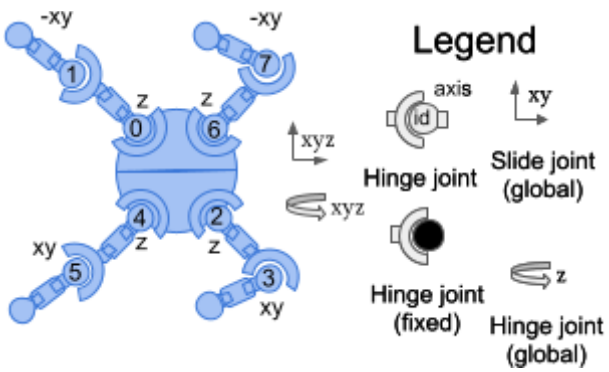


Figure 14 : Schéma cinématique du robot Ant

Notre idée a été de reprendre cet exemple de robot simple et d'en faire une modélisation qui soit rapide et facile à prototyper. Notre robot nommé Spider reprend le même modèle mécanique que Ant, avec cependant une différence : chaque patte possède un degré de liberté supplémentaire. Nous avons ce fait ce choix par curiosité, tout en sachant qu'il nous était facile de prototyper un tel système grâce aux servomoteurs à notre disposition au labo.

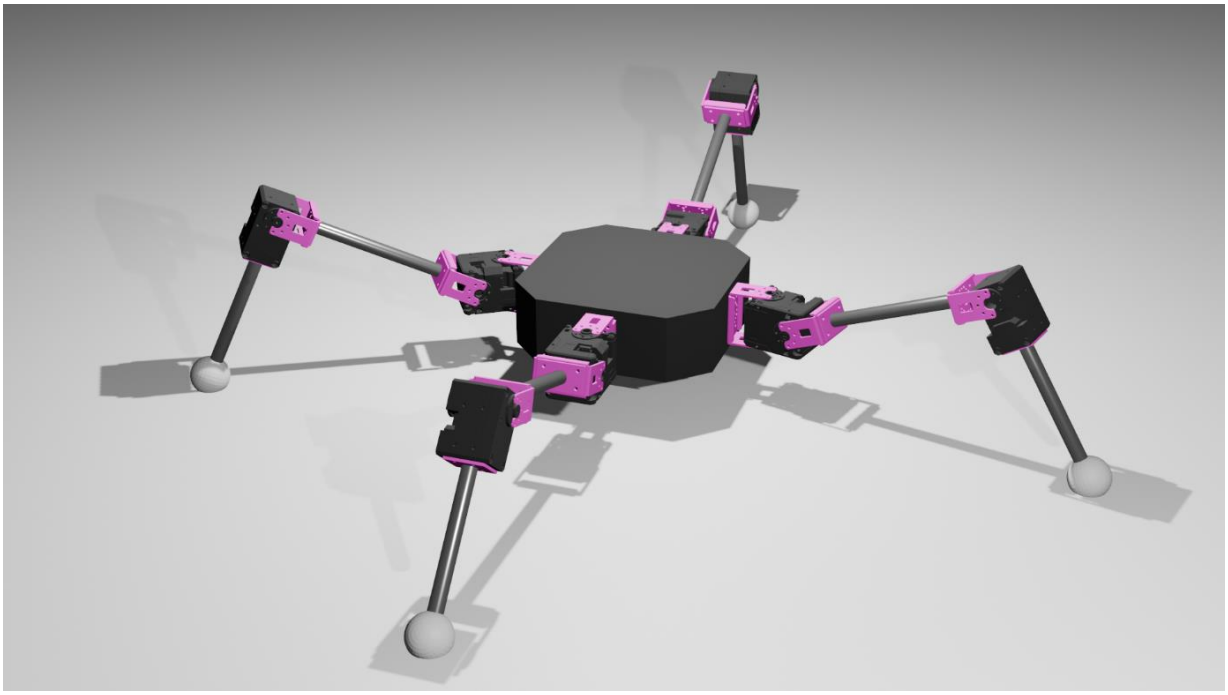


Figure 15 : Vue d'artiste de Spider (modélisation réalisée sur Blender)

2.2.3.2 Modélisation

Une fois le système conceptualisé, il nous a fallu créer un modèle sur Isaac Sim au format USD. En nous servant des servomoteurs modélisés plus tôt, nous avons scripté un modèle de l'araignée en python. La modélisation se décompose en 4 grandes étapes : dans un premier temps il faut placer les membres du corps. On commence par le corps central, puis on rajoute les avant-bras et les bras. La deuxième étape consiste à placer les servomoteurs qui embarquent des joints de rotation. Ensuite il faut rajouter des joints de fixation pour relier physiquement les membres du corps avec les moteurs. Cette étape est fastidieuse car il faut aligner la position et l'orientation désirées du corps 1 de la fixation dans le référentiel du corps 0.

Enfin, la dernière étape consiste à enregistrer le fichier USD de manière à ce qu'il soit facilement importable dans les scénarios d'apprentissage d'Isaac Lab. Pour cela il y a trois éléments à garder en tête. D'abord il faut déclarer la primitive principale du robot comme *DefaultPrim*. Les fichiers USD sont des descriptions de scènes avec potentiellement plusieurs objets. Si on veut qu'Isaac Lab sache comment trouver le robot dans la scène il faut marquer ce robot comme sujet principal de la scène. Ensuite on doit autoriser Isaac Sim à considérer le robot comme un système cohérent composé de corps et de joints, afin d'avoir accès à l'API *articulations* et toutes les commodités qui l'accompagnent, comme le contrôle des joints, les données de vitesse et de position des différents membres … Pour cela il suffit de donner à la primitive principale la propriété *articulationRoot*. Enfin, dans le cas où le modèle utilise des

meshs customisés (c'est le cas en l'occurrence des servomoteurs qu'on a modélisé), il faut sauvegarder le fichier USD avec l'option *flattened*. Cela va créer une copie des meshs à l'intérieur du fichier, évitant ainsi les éventuels bugs de chemin d'accès au fichier 3D.

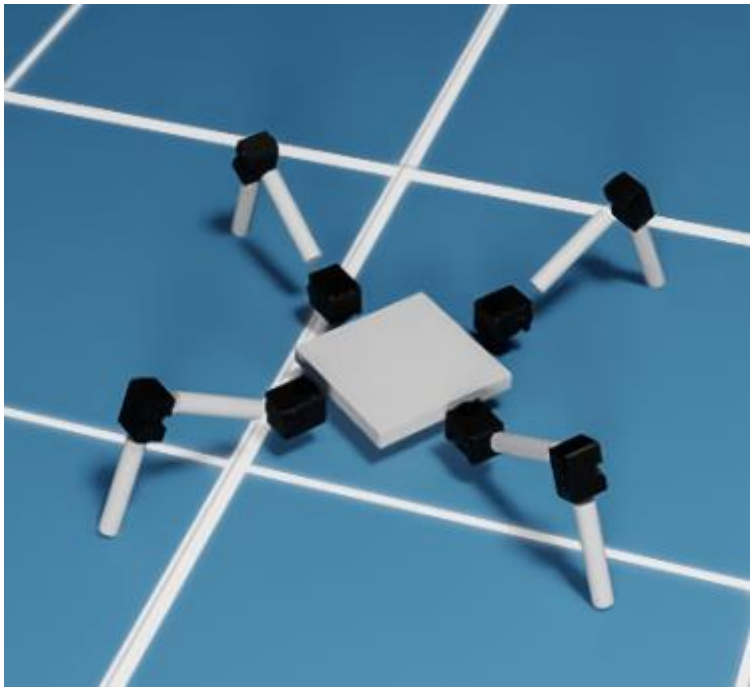


Figure 16 : Modèle simulé de Spider

Afin de gagner du temps sur cette phase de modélisation, nous n'avons pas connecté visuellement les différentes parties du robot. Elles sont seulement connectées physiquement. On peut voir dans la 16 que les pattes ont l'air de flotter. Cette simplification ne diminue pas la qualité de la simulation car les différents éléments sont positionnés avec précision. Rendre le modèle du robot plus léger pour l'ordinateur (c'est-à-dire avec le moins de vertex possible) améliore la vitesse de calcul et les rendements graphiques lors de l'apprentissage.

2.2.3.3 Fonctions de déplacement « à la main »

Une fois le robot modélisé, nous avons voulu tester le système en codant des commandes moteur. Nous avons programmé des valeurs de position pour chaque joint à des intervalles de temps donnés. Empiriquement, nous avons réussi à donner à l'araignée un mouvement assez mécanique et saccadé qui lui permet néanmoins de ramper sur le sol et de se déplacer. Nous avons vite conclu que ce procédé était long et fastidieux. L'intérêt est donc de voir si un Spider entraîné en simulation peut réaliser une meilleure performance que le robot qui se déplace « manuellement ».

2.2.4 Conception de spider

Dans l'optique de faire du « Sim2Real », nous avons réalisé tout le travail de prototypage du robot. L'enjeu ici était de réaliser un modèle réel du robot qui soit à la fois fiable, fidèle à la simulation, solide pour encaisser d'éventuels mouvements brusques et rapide à monter.

2.2.4.1 Dimensionnement mécanique

Le robot possède quatre pattes dont on détaille plus loin la conception et un corps central. Pour le corps nous avons pris une plaque en bois de 15cm par 15cm. L'intérêt du bois est qu'il est facile à trouer, ce qui simplifie la fixation des différentes composantes (pattes, ordinateur de bord…) à l'aide de vis à bois. Le désavantage de cette plaque est qu'elle est très épaisse et donc relativement lourde. Une des possibles améliorations de la structure mécanique du robot serait de remplacer cette plaque par un corps customisé en impression 3D par exemple, le PLA étant assez léger.

Les pattes mesurent chacune 33cm (nous verrons plus tard qu'on a raccourci de 8cm la première section des pattes pour une question de répartition des efforts mécaniques).

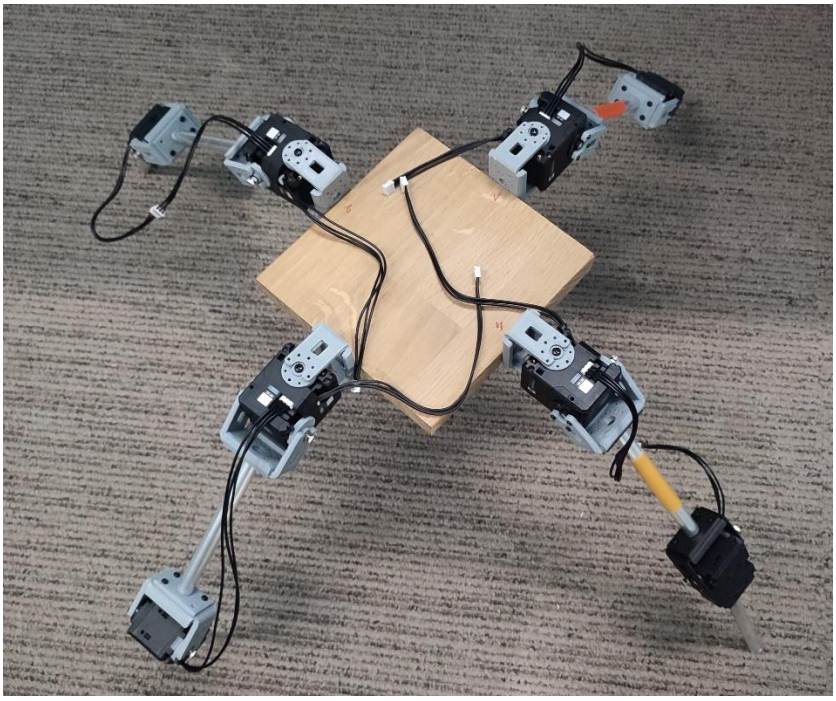


Figure 17 : Structure mécanique de Spider

2.2.4.2 Les servomoteurs

Les servomoteurs Dynamixel de chez ROBOTIS correspondent parfaitement à nos besoins. Ils sont à la fois robustes, sécurisés (il y a un failsafe mode en cas de surcharge de couple sur le rotor), et il existe dans la communauté un bon nombre de packages qui fournissent des surcouches logicielles adéquates pour une utilisation rapide du moteur. Ils sont fournis avec une petite carte électronique qui sert d'interface USB avec l'ordinateur, et le SDK de chez ROBOTIS assure toutes les fonctions de bas niveau qui permettent de contrôler, paramétrer et monitorer les moteurs.

Pour un paramétrage facile et rapide, il est possible d'utiliser le logiciel Dynamixel Wizard fourni par le fabricant. Parmi les paramètres importants on compte l'ID du moteur, ce qui est utile lorsqu'on utilise plusieurs moteurs dans le même robot, les gains du contrôleur PID, le mode de contrôle (PWM, position, vitesse) ou encore les limites physiques positionnelles du moteur.

Les moteurs Dynamixel peuvent se brancher entre eux en série ce qui est parfait pour construire des bras robotiques en limitant les câbles longs. Dans le cas de Spider, nous avons connecté quatre séries de 2 moteurs. Le premier servomoteur de la série est un 2XL430. C'est un moteur avec 2 axes. Le second est un XL430, il fournit un seul degré de liberté à la patte.

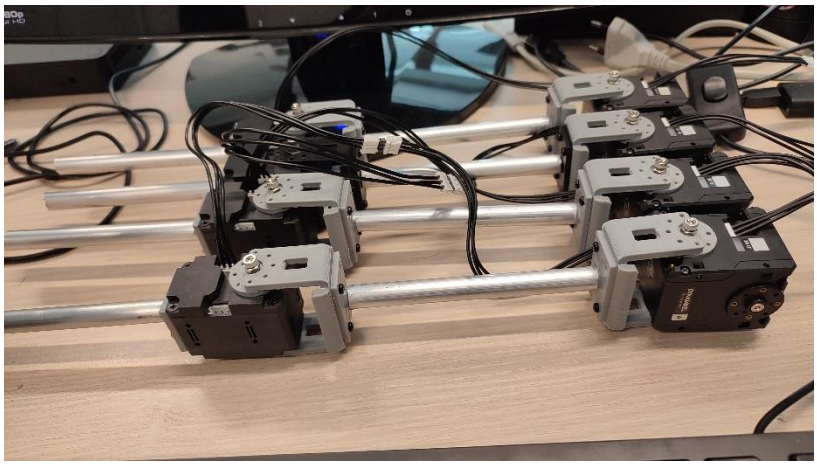
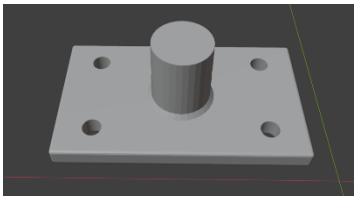


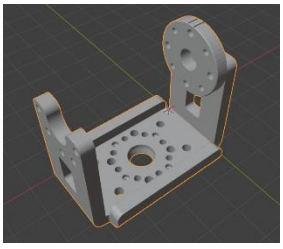
Figure 18 : Les 4 pattes de Spider

2.2.4.3 CAO et Impressions 3D

Toujours dans l'idée d'un prototypage rapide et afin d'éviter les temps d'attente de livraisons de pièces spécifiques, nous avons utilisé l'imprimante 3D du CNES. Pour relier mécaniquement les servomoteurs entre eux et au torse de Spider, nous avons modélisé des pièces adaptées aux rotors des moteurs, et des plaques avec un cylindre pour les insérer dans des profilés creux en aluminium.

Plusieurs impressions ont servi d'essai avant d'arriver aux modèles finaux (voir ci-dessous). Certaines pièces par exemple ont été épaissies par question de solidité, les premières ayant cassé.





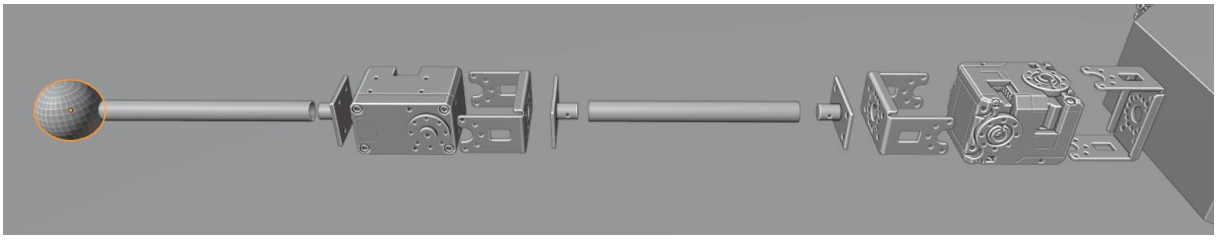


Figure 19 : Vue éclatée d'une patte dans Blender

Les imprimantes 3D ont la capacité d'imprimer avec différents filaments de matière. Certains ont des propriétés physiques telles que la résistance à la chaleur, la solidité mécanique, l'électro dissipation… En vue de nos besoins nous nous sommes contentés du filament plastique le plus basique, le PLA. Mais pour s'assurer un minimum de solidité, nous avons conçu des pièces suffisamment épaisses pour résister au poids total de l'araignée.

2.2.4.4 Structure électronique

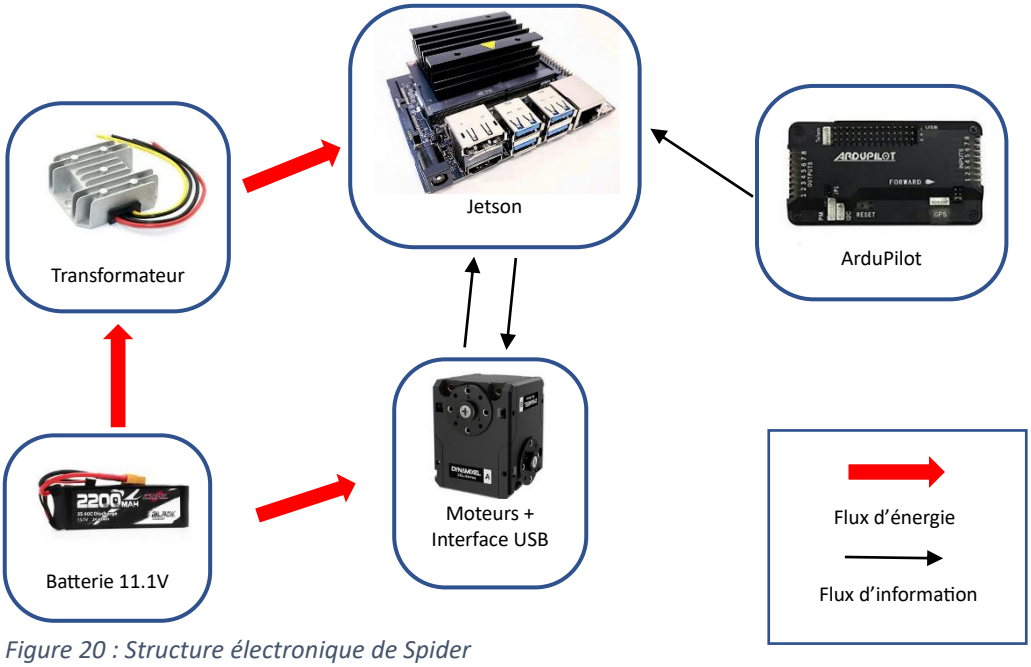


Figure 20 : Structure électronique de Spider

Le cœur électronique du robot est son ordinateur de bord : nous avons choisi une Jetson Nano Developer Kit [12]. C'est une carte embarquée conçue par Nvidia. Elle a l'avantage de posséder un GPU ce qui est parfait pour du Machine Learning embarqué. En revanche, pour des raisons de sécurité elle possède un failsafe qui se déclenche lorsque le courant ou la tension d'alimentation ne sont pas précisément dans la fourchette demandée par la documentation. Cela implique de dimensionner correctement les batteries. La carte accepte une tension d'alimentation de 5V et limite le courant à 4A.

La batterie est une Lipo 3S. Elle alimente à la fois les servomoteurs et la Jetson via un transformateur 5V. Provisoirement, par manque de matériel électronique, on a alimenté les batteries avec la Lipo 3S et la Jetson avec une prise secteur. Dans le futur une amélioration de ce système électronique permettra à Spider d'être totalement autonome et de ne pas avoir à rester attaché à un câble.

Actuellement le seul capteur utilisé est l'IMU d'une ArduPilot. Elle envoie à la Jetson à un rythme soutenu (environ 10 fois par secondes) les informations du Gyroscope, accéléromètre ainsi que les vitesses et les positions angulaires du corps principal.

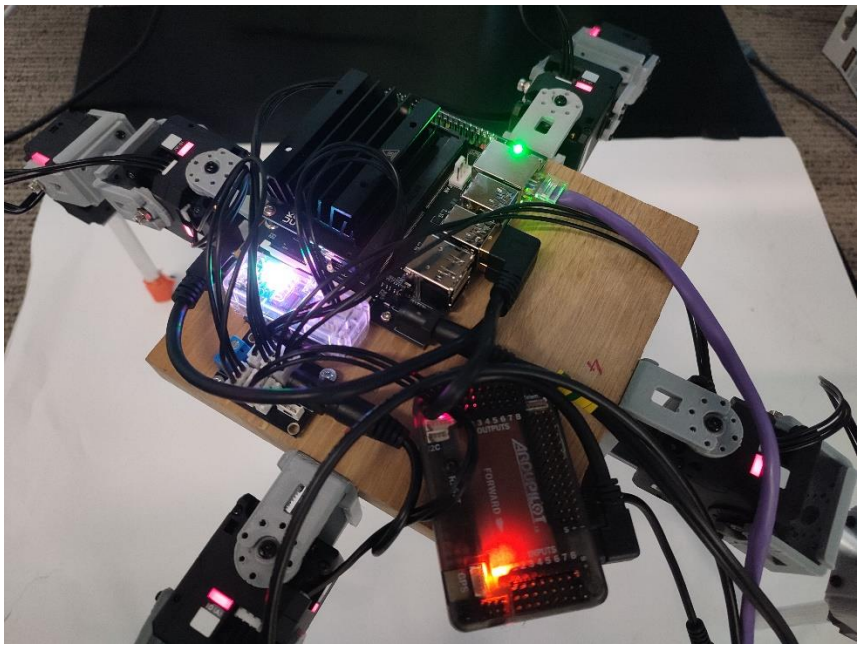


Figure 21 : Image de l'électronique embarquée de Spider

2.2.4.5 Structure logicielle

La structure logicielle est relativement simple. Le framework ROS2 est très adapté pour faire de la communication d'informations sur un système embarqué en temps réel. Le package ROS2 contient trois Nodes : un pour interfacer les commandes avec les moteurs, un pour réceptionner et partager les données de l'IMU et un pour former le vecteur des observations du robot, le mettre en entrée du réseau de neurones (donc de la politique) et envoyer la sortie au Node moteur.

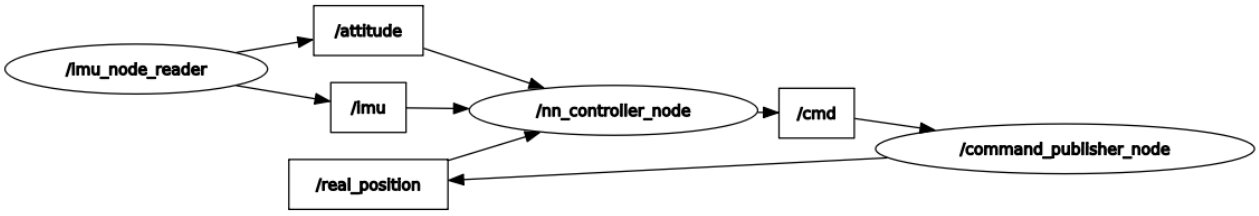


Figure 22 : Graphe des nodes ROS

2.3 Apprentissage

L'apprentissage en simulation est une phase critique du projet car la réalisation d'une « policy » efficace aussi bien dans la simulation que dans le système réel constitue l'un des enjeux de ce stage. Il existe trois grandes branches de l'apprentissage machine : l'apprentissage supervisé, l'apprentissage non supervisé et l'apprentissage par renforcement. C'est cette dernière méthode qui nous intéresse. Elle consiste à placer un agent dans son environnement et le laisser interagir avec ce dernier. Il n'y a pas de données labellisées, le système crée ses propres données en réalisant des épisodes. Par essais et erreurs, le système

comprend de lui-même comment gagner des points selon une fonction de récompense définie par l'ingénieur. Comme souvent, cette méthode d'apprentissage repose donc sur l'optimisation d'une fonction.

2.3.1 Apprentissage par renforcement

L'apprentissage par renforcement est une branche de l'intelligence artificielle où un agent apprend à prendre des décisions en interagissant avec son environnement. Contrairement à d'autres formes d'apprentissage supervisé, il n'a pas besoin de données étiquetées. Il reçoit plutôt des récompenses, positives ou négatives, en fonction de ses actions et cherche à maximiser son gain total.

> *Reinforcement learning is a framework for solving control tasks (also called decision problems) by building agents that learn from the environment by interacting with it through trial and error and receiving rewards (positive or negative) as unique feedback.*

Définition formelle de l'apprentissage par renforcement – HuggingFace [13]

Le concept repose sur les composantes suivantes :

- Agent : l'agent est le système qui prend les décisions. Il effectue des actions en se basant sur sa perception de l'environnement dans lequel il se trouve. Il est à noter qu'il ne perçoit qu'une partie de l'environnement et qu'il n'a pas forcément accès à toutes les informations du monde dans lequel il évolue.
- Environnement : l'environnement représente le contexte d'apprentissage de l'agent. Cela correspond à l'ensemble des situations ou configurations dans lesquelles il peut se retrouver. L'environnement change en réponse aux actions de l'agent.
- État : l'état est la représentation de la situation actuelle dans laquelle se trouve l'agent.
- Observation : l'observation constitue l'ensemble des informations perçues par l'agent. C'est un sous-ensemble de l'état. En robotique ces informations parviennent au robot grâce à ses capteurs.
- Action : les actions sont les décisions prises par l'agent à chaque étape. Chaque action influence l'environnement, en modifiant son état, et peut entraîner des récompenses positives ou négatives.
- Récompense : la récompense est un retour que reçoit l'agent après avoir effectué une action dans un certain état. Elle peut être positive ou négative. Le but de l'agent est de maximiser la somme des récompenses au fil du temps. L'utilisateur définit une fonction de récompenses qui aide l'agent à comprendre quels états sont plus ou moins favorables.
- Politique : la politique est la stratégie de l'agent qui détermine quelles actions il doit entreprendre dans chaque état. Elle peut être stochastique, avec des actions choisies de manière probabiliste, ou déterministe, une seule action étant alors possible pour chaque état.

Pour formaliser ces données, on note :

- S : l'espace des états
- A : l'espace des actions
- $P(s' | s, a)$: la dynamique de transition
- $R(s, a)$: la fonction de « reward » (récompense en anglais)

- $\pi(a \mid s)$: la politique, ou « policy » de l'agent. C'est une fonction qui prend l'état en entrée et qui donne une action en sortie.
- γ : le facteur d'actualisation qui pondère l'importance des récompenses futures $(0 \leq \gamma \leq 1)$

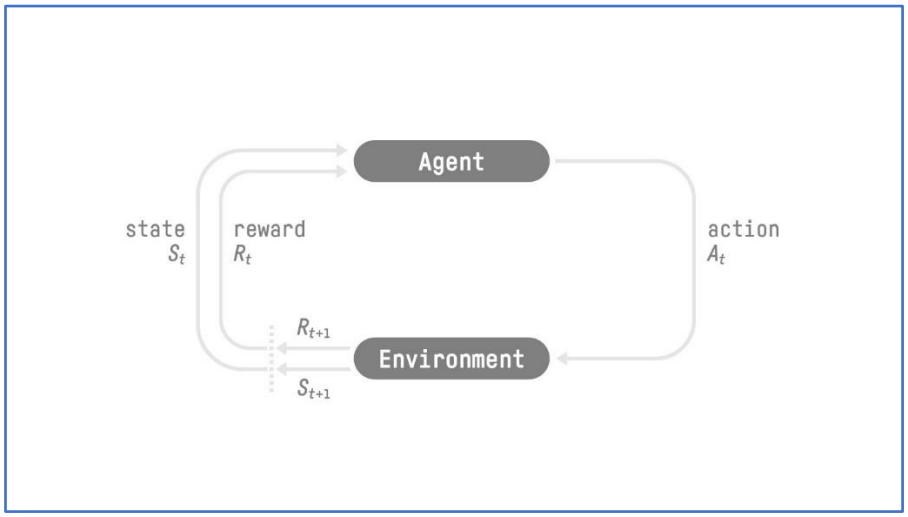


Figure 23 : Boucle d'état, d'action, de récompense et d'état suivant

Les domaines d'application de cette méthode d'apprentissage machine incluent des jeux complexes (comme AlphaGo, StarCraft…) ou plus simples (comme tous les jeux ATARI), la Robotique, la gestion de ressources ou encore la finance algorithmique.

L'apprentissage par renforcement est largement utilisé en robotique pour enseigner aux agents comment accomplir des tâches complexes, telles que la manipulation d'objets, la planification de trajectoire et la navigation dans des environnements. En robotique, les environnements sont souvent dynamiques, imprévisibles et peuvent être compliqués à modéliser mathématiquement. L'apprentissage par renforcement est donc une technique particulièrement adaptée car l'agent robotique peut explorer l'environnement et découvrir quelles actions mènent aux meilleures récompenses, même dans des environnements changeants. Il n'y a pas besoin de données annotées à l'avance. Le robot apprend par essais et par erreurs, ce qui est utile lorsque le modèle de l'environnement est partiel ou inconnu.

Enfin, le robot apprend à maximiser ses récompenses à long terme, ce qui le pousse à optimiser ses performances dans le temps. Cependant, définir les « bonnes » récompenses n'est généralement pas évident. Ce processus, connu sous le nom de « reward engineering », comporte le risque d'affiner de manière excessive les récompenses, ce qui peut parfois biaiser l'apprentissage. En effet, un excès de reward engineering peut amener le robot à adopter des comportements inattendus pour accumuler des récompenses définies, plutôt que de véritablement accomplir la tâche souhaitée. Définir des fonctions de rewards implique de savoir à l'avance quel résultat on s'attend à obtenir, c'est-à-dire vers quelle politique on veut converger. C'est à l'agent d'apprendre comment converger vers ces résultats.

2.3.2 État de l'art

2.3.2.1 Processus Décisionnel de Markov (MDP)

Les MDP sont une formalisation classique de la prise de décision séquentielle. C'est une manière de modéliser les problèmes d'optimisation tels que celui de l'apprentissage par renforcement. Un MDP est défini par un quintuplet : $M = (S, A, P, R, \gamma)$. La figure 23 représente bien ce principe : un agent se base sur l'état de l'environnement et sur sa récompense à l'étape précédente pour choisir une nouvelle action qui va à nouveau influencer l'environnement.

L'hypothèse de Markov dans un MDP induit que l'état s_t contient toute l'information nécessaire pour déterminer la distribution de l'état futur s_{t+1}. Cette propriété s'écrit formellement :

$$P(s_{t+1}, r_{t+1} | s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0) = P(s_{t+1}, r_{t+1} \mid s_t, a_t)$$

L'objectif de cet agent et donc la résolution du problème d'apprentissage par renforcement, c'est-à-dire trouver une politique π^* qui maximise la récompense cumulée attendue :

$$G_t(\pi) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$$

Dans la littérature on classifie les méthodes d'apprentissage par renforcement en deux grandes catégories :

- Les méthodes basées sur la politique. L'agent apprend directement quelle action il doit entreprendre en fonction de l'état actuel de l'environnement.
- Les méthodes basées sur la valeur. Cette fois on apprend à l'agent à deviner quel état lui rapporte le plus de bénéfices. Ensuite il prend les décisions nécessaires pour arriver à cet état.

2.3.2.2 Méthode à base de valeur

Les méthodes à base de valeur visent à approximer la fonction de valeur optimale afin de déduire une politique optimale. L'agent apprend à reconnaître quel état lui est bénéfique à l'instar des états qui lui rapportent moins de points.

La fonction de valeur qu'il faut optimiser est :

$$v_\pi(s) = \mathrm{E}_\pi[G_t \mid S_t = s]$$

Cette fonction signifie que la fonction de valeur d'état (v_π) donnera la valeur attendue si l'agent commence à l'état s et suit ensuite la politique π.

Pour éviter de devoir calculer chaque valeur comme la somme du rendu attendu, ce qui peut être long, on utilise l'équation de Bellman pour calculer la valeur comme étant la somme de la récompense immédiate + la valeur de l'état suivant :

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma \cdot v(S_{t+1}) \mid S_t = s]$$

Une méthode à base de valeur qui est largement utilisée est celle du Q-Learning. Le principe du Q-Learning est d'entrainer une fonction Q (une fonction de valeur d'état) qui en fait est une Q-table (Q pour Qualité) contenant toutes les valeurs des paires état/action. Donc pour une

paire état/action donnée, la fonction Q recherche dans sa table la valeur correspondante. Une fois l'entraînement terminé, on obtient une fonction optimale Q^*. La connaissance d'une telle fonction nous donne la politique optimale π^* de par l'équation suivante :

$$\pi^*(s) = \arg\max_a Q^*(s, a)$$

Pour entrainer cette fonction Q, on commence par l'initialiser avec des valeurs toutes égales à 0. Ensuite on utilise une stratégie « epsilon-greedy », c'est-à-dire une stratégie qui va choisir avec une probabilité de ε une action random, et avec une probabilité de $1 - \varepsilon$ l'action qui lui rapportera le plus de points selon la table actuelle. Au début de l'apprentissage on choisit $\varepsilon = 1.0$ car on veut que l'agent explore des politiques. Et petit à petit on réduit cette valeur pour favoriser l'utilisation de la table. C'est ce qu'on appelle le compromis entre exploitation et exploration. La dernière étape après le choix d'une action A_t est la mise à jour de la table Q selon :

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

Cette méthode possède une évolution naturelle vers l'implémentation d'un réseau de neurones. Ça s'appelle le « Deep Q-Learning » (ou DQN pour Deep Q-Network) et consiste à entraîner un réseau de neurones $Q_\theta(s, a)$ pour qu'il approxime la table optimale $Q^*(s, a)$.

2.3.2.3 Méthode à base de politique : Policy Gradient

Si la méthode Q-Learning est très efficace et converge presque toujours vers la politique optimale, elle n'est pas adaptée pour des applications robotiques. En effet l'espace des états et des actions est continu, quasi-infini. Par exemple Spider peut choisir pour chacun de ses 12 joints une position parmi 4096 disponibles. Le nombre de combinaisons est beaucoup trop grand pour espérer mettre tout cela dans une table. C'est là qu'interviennent les méthodes « Policy Gradient ».

Plutôt que d'entraîner une table de valeurs en fonction d'une paire état/action, on entraîne directement une politique stochastique π_θ sous forme d'un réseau de neurones. Ce réseau nous renvoie une probabilité de distribution des actions.

$$\pi_\theta(s) = \mathbb{P}[A|s; \theta]$$

Pour évaluer la performance de cette politique on a besoin d'une fonction objectif à optimiser :

$$J(\theta) = E_{\tau \sim \pi}[G_t(\pi)]$$

Où τ représente la trajectoire, c'est-à-dire la séquence d'états et d'actions au cours d'un épisode. Cette fonction objectif est donc égale à la moyenne des valeurs d'état attendue pour une trajectoire donnée.

Pour mettre à jour la politique, on performe un algorithme de « gradient ascent ». Il nécessite de dériver la fonction objectif, ce qui est très long car il faudrait calculer la probabilité de chaque trajectoire possible. On utilise donc le théorème Policy Gradient pour reformuler la fonction objectif en une fonction différentiable qui ne fait pas intervenir la dérivation de la distribution d'état.

$$\nabla_\theta J(\pi_\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(A_t|S_t) \cdot G_t]$$

Les méthodes basées sur la politique sont avantageuses car elles sont simples à intégrer, il n'y a qu'un réseau de neurones à stocker et pas de table de valeurs. On s'affranchit du compromis exploitation/exploration et des éventuels problèmes d'aliasing de perception qui interviennent lorsque deux observations similaires sont en fait deux états différents et requièrent donc deux actions différentes.

Cependant il y a aussi des désavantages par rapport au Q-Learning. Les méthodes policy gradient convergent régulièrement vers des optimums locaux et non globaux et l'apprentissage étape par étape est plus long.

2.3.2.4 Méthodes Actor-Critic

Ce sont des méthodes qui combinent une politique à optimiser π_θ et une estimation de la valeur d'état $Q(s,a)$ pour guider l'agent. L'agent décide, via sa politique actuelle d'une action (actor) et on compare cette action à une estimation Q de la valeur (critic).

L'action performée dans l'environnement renvoie un nouveau couple state/reward S_{t+1} et R_{t+1} qui sont redonnés à l'actor et au critic pour l'itération suivante. Puis on met à jour la politique de l'actor à l'aide de l'estimation du critic.

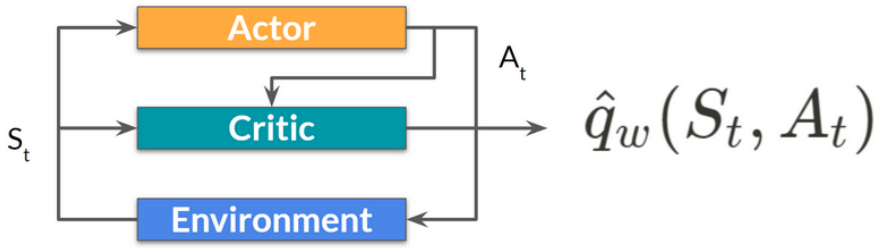


Figure 24 : Procédé Actor - Critic

On change les poids du réseau de la politique avec :

$$\Delta\theta = \alpha\nabla_\theta\big(log\pi_\theta(s,a)\big)\hat{q}_w(s,a)$$

Une méthode Actor-Critic assez répandue est la méthode A2C. Elle introduit la fonction avantage :

$$A(s,a) = Q(s,a) - V(s)$$

qui mesure à quel point une action est meilleure ou pire que la moyenne (ici V est la valeur moyenne à l'état s).

2.3.2.5 Proximal Policy Optimization (PPO)

PPO est une méthode Actor-Critic largement adoptée en robotique pour sa simplicité et sa robustesse. Son objectif est d'éviter les mises à jour trop agressives qui déstabilisent l'apprentissage. PPO est particulièrement adaptée aux tâches longues et aux actions parallèles, ce qui accélère le processus d'entraînement.

Dans les grandes lignes, on introduit un ratio de variation de la politique pour une étape donnée :

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

et on redéfinit une fonction objectif clippée :

$$L^{CLIP}(\theta) = \mathbb{E}_t[\min(r_t(\theta) \cdot A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon) \cdot A_t]$$

[14], [15], [16], [17]

2.3.3 Création d'environnement dans Isaac Lab

Les environnements d'apprentissage sont des cadres dans lesquels un agent peut interagir avec son environnement pour apprendre des tâches spécifiques en utilisant l'apprentissage par renforcement. Ces environnements modélisent le monde réel ou des simulations réalistes, permettant aux agents d'apprendre à résoudre des problèmes tout en minimisant les risques et coûts associés aux tests physiques réels. Ils permettent également de répéter les expériences rapidement et à grande échelle, ce qui accélère le processus d'apprentissage. Dans le domaine de la robotique, les environnements d'apprentissage sont essentiels pour simuler des scénarios variés, tester différents algorithmes de contrôle et améliorer les performances des robots avant leur déploiement en conditions réelles. Ces environnements offrent une flexibilité nécessaire pour explorer une large gamme de stratégies et affiner les modèles. Ayant réalisé la partie de simulation du projet en utilisant Isaac Sim, on s'est naturellement tourné vers la plateforme de création d'environnements d'apprentissage qui découle d'Isaac Sim : Isaac Lab [3].

Isaac Lab est une extension d'Isaac Sim, conçue pour offrir un environnement de simulation immersif et réaliste, particulièrement adapté à la recherche et au développement de robots autonomes. Isaac Lab se distingue par sa capacité à intégrer des technologies avancées telles que la simulation physique précise, la vision par ordinateur et l'interaction robotique dans des environnements complexes. Isaac Lab permet de créer des environnements de simulation avec une grande fidélité physique et visuelle, ce qui est essentiel pour entraîner des robots dans des conditions proches de la réalité. Il prend aussi en charge la modélisation de capteurs et l'intégration de frameworks de robotique tels que ROS2. Il est optimisé pour fonctionner avec des algorithmes d'apprentissage par renforcement. En effet, il embarque des bibliothèques telles que Stable Baselines 3 [18], RL_Games [19] et rsl_rl [20], celui qu'on utilisera en majorité.

2.3.4 Apprentissage de Spider

2.3.4.1 Création de l'environnement

La première mission de Spider est de se déplacer en ligne droite en utilisant ses pattes tel une araignée. Pour cela nous avons commencé par préparer l'environnement d'apprentissage. Concrètement, on réutilise le modèle simulé de l'araignée dans l'environnement. On crée ensuite deux fichiers de configuration : un fichier pour définir avec précision l'environnement avec l'ensemble des fonctions de reward, d'observation, d'action… Et un autre fichier qui définit l'agent à entraîner. Ce fichier contient les hyper-paramètres tels que la taille du réseau de neurones ou γ, la valeur du discount dans la fonction de valeur.

Dans le cas de Spider : le réseau de neurones contient un input layer de dimension 48 qui correspond à la taille du vecteur d'observation que l'on fournit à l'agent. Puis on a 3 hidden layers de dimensions 400, 200 et 100 (le même réseau que pour l'exemple Ant) et un output layer de dimension 12, une dimension par joint du robot.

Isaac Lab fournit les fichiers qui exécutent l'entraînement. Pour pouvoir s'en servir il faut seulement « enregistrer » l'environnement que l'on a configuré dans la liste des environnements d'Isaac Lab. Cela étant, Isaac Lab va instancier par défaut 4096 de ces environnements simultanément. C'est l'avantage de l'apprentissage en simulation, on peut entraîner plusieurs milliers d'agents en même temps ce qui améliore considérablement la performance des algorithmes d'apprentissage.

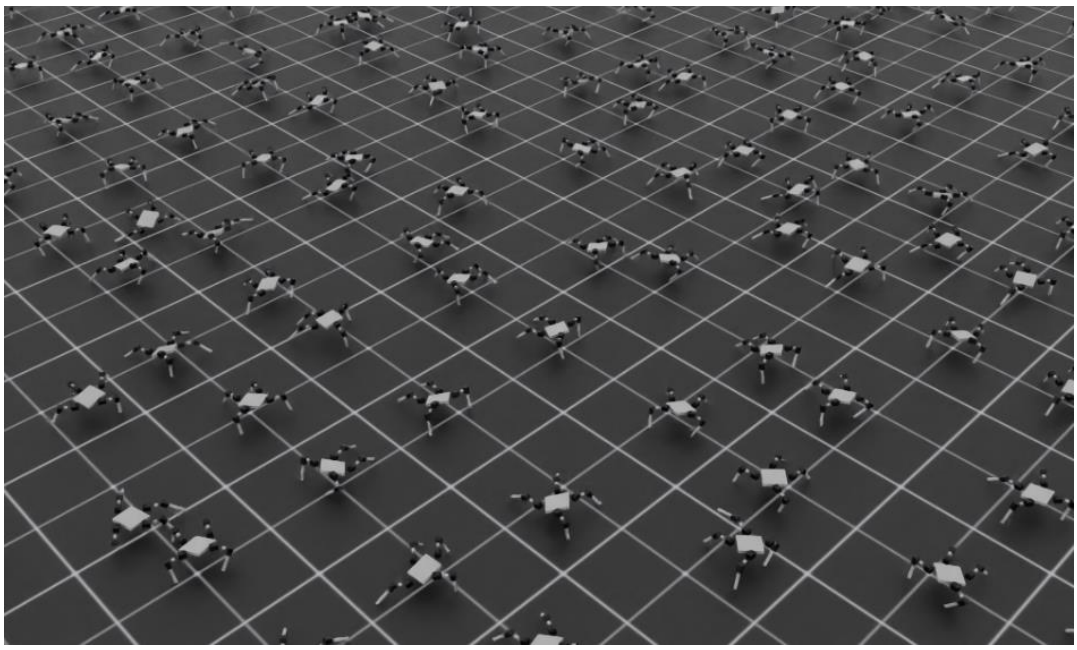


Figure 25 : Environnements d'apprentissage

En plus de toutes les contraintes physiques, il est possible de rajouter des règles pour favoriser l'apprentissage. On commence par définir l'espace des observations. C'est-à-dire que l'on fixe les données auxquelles l'agent a accès, aussi bien pendant l'entraînement que pendant une éventuelle mission. Cet espace des observations correspond à l'Input du réseau de neurones.

Si dans certains exemples d'application d'apprentissage par renforcement il est possible de donner l'entièreté de l'état à l'agent durant l'entraînement (comme par exemple dans la plupart des jeux ATARI, au jeu d'échec, …) nos environnements contiennent bien trop d'information physiques pour que ce soit intéressant. C'est pourquoi on sélectionne les données physiques qui nous semblent pertinentes. Dans le cas de Spider, l'agent a accès à :

- La position et la vitesse de son torse dans l'espace (en trois dimensions)
- Sa vitesse angulaire,
- Son roulis et son lacet,
- La différence angulaire entre son cap actuel et son cap désiré

- La projection de son axe z local sur l'axe z du monde,
- La position et la vitesse de ses articulations
- Et son action (i.e. sa décision) à l'étape précédente.

Par la suite on peut définir des conditions de « reset » (réinitialisation) des environnements. Ces conditions servent à « annuler » un agent qui est manifestement mal engagé dans son épisode et dont on sait déjà qu'il ne va rien apporter à l'apprentissage. Un tel agent qui vérifie une condition de reset est supprimé, puis son environnement est réinitialisé et un nouvel épisode commence avec un nouvel agent. Dans le cas de Spider on a décidé de réinitialiser les agents dont le torse est soit trop haut ou trop bas (on a défini des limites dans l'axe z). Un robot qui descend trop va frotter contre le sol ce qui est mauvais car nous avons placé les batteries et de l'électronique en-dessous du ventre du robot et la limite haute permet d'éviter de laisser émerger des comportements où Spider saute, car même si une telle politique peut mathématiquement gagner des points, nous avons souhaité imiter le déplacement d'un arachnide rampant.

2.3.4.2 Calcul des Rewards

La définition de la fonction de reward joue un rôle critique dans l'apprentissage. Elle constitue le seul retour que l'agent obtient sur son comportement. Voici celles qu'on a utilisées pour Spider :

- Heading Reward : le robot est récompensé s'il suit bien le cap désiré

$$Rew_{Heading} = \begin{cases} 1 & Si\ p_{cap \to cap\ désiré} > 0.8 \\ p_{cap \to cap\ désiré} & Sinon \end{cases}$$

- Up Reward : on récompense un robot qui garde son torse horizontal

$$Rew_{Up} = \begin{cases} up_{weight} & Si\ z_{Spider} \cdot z_{world} > 0.93 \\ 0 & Sinon \end{cases}$$

- Energy penalty : on enlève des points proportionnellement à l'amplitude de mouvement (qui requiert naturellement plus d'énergie). On définit deux pénalités différentes :
  - $pen_{electricity} = actions \times joint_{velocity}$
  - $pen_{actions} = actions \times actions$
- On récompense un agent qui réussit à se rapprocher de son objectif, c'est la progress reward, elle se base sur la distance parcourue par rapport à la distance qu'on lui demande de parcourir.
- Et enfin on applique un malus constant à un agent qui est réinitialisé.

La difficulté réside non seulement dans la définition de la fonction de récompense mais aussi dans la pondération de ses différentes composantes. C'est ce qu'on voit dans la partie suivante.

2.3.4.3 Tuning des rewards et des paramètres d'apprentissage

Pour pondérer les composantes de la fonction de reward, il n'y a pas de méthode prédéterminée. Il faut faire des essais et petit à petit essayer d'arriver à un résultat stable. Cette pondération a toute une importance car si par exemple on surévalue l'Up reward au détriment de la Progress reward, on se retrouve avec un Spider parfaitement horizontal et stable mais qui n'avance pas.

Et chaque paramètre a son rôle à jouer. Sur Spider, nous avons essayé de combiner les rewards Progress, Alive, Up et Heading et nous avons obtenu un agent qui parvient à se déplacer sensiblement (ou plutôt à basculer) dans la bonne direction. Mais dès que nous avons augmenté un peu la mobilité des joints, nous nous sommes retrouvés avec un agent qui reste sur place en agitant les pattes de manière aléatoire. A l'inverse, en mettant un gain trop important sur $pen_{electricity}$ nous avons obtenu un robot qui replie ses pattes sur lui-même et garde une position assez neutre. Il parvient ainsi à limiter ses mouvements et ne pas être pénalisé, mais il ne gagne aucun point relatif à sa mission principale.

Pour faciliter l'étude de l'évolution des apprentissages on peut utiliser Tensorboard [21], un outil qui récupère individuellement chaque composante de la reward et les affiche dans une visualisation web.

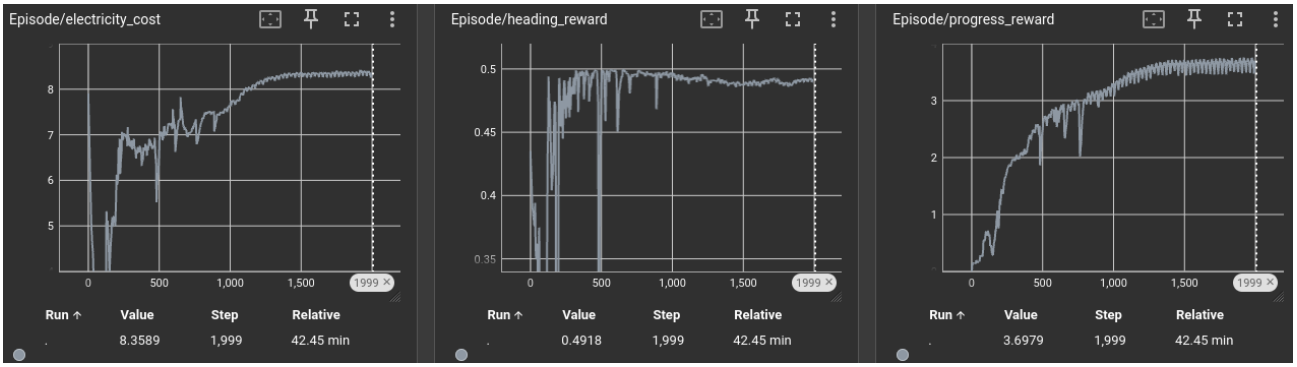


Figure 26 : Rewards Pen electricity (à gauche), Heading (au milieu) et Progress (à droite)

Voici dans la figure 26 un exemple de résultat. Cet apprentissage a donné un agent qui parvient à agiter les pattes de manière relativement coordonnée pour se propulser dans la bonne direction. Très vite la reward Heading monte et finit par stagner à 0,5 par épisode. L'agent comprend très vite qu'il gagne à suivre la bonne direction. Progress monte aussi mais plus lentement, et finit aussi par stagner vers 3,5. Peu à peu l'agent apprend à se propulser de manière plus efficace et plus rapide, mais il est limité par la durée maximale d'un épisode, sans quoi il pourrait gagner probablement beaucoup plus de points. Par ailleurs on voit que la pénalité de mouvement augmente aussi. Cela signifie que malgré la pénalité, le robot trouve un compromis quantité de mouvement/mission et accepte de perdre des points si c'est pour en gagner plus par ailleurs. Cette pénalité sert aussi à aider l'agent à trouver un mouvement coordonné car elle limite tout de même les mouvements amples et aléatoires.

Le meilleur entraînement que nous avons obtenu pour Spider montre un robot qui réussit à courir dans la bonne direction. Qualitativement, son mouvement ressemble plus à un cheval au galop qui décolle ses quatre pattes pour avancer, qu'à une araignée qui décolle les pattes

une par une. Peu importe les contraintes physiques, l'agent réussit à trouver une manière de marcher même si ce n'est pas tout à fait celle à laquelle on s'attend.

Pour aller plus loin nous avons fait des expériences sur les paramètres physiques de l'environnement pendant l'entraînement. Nous avons notamment fait varier la valeur de la constante de gravitation g. En soumettant par exemple le robot à une gravité lunaire, nous observons un agent avec une toute nouvelle méthode de déplacement : à l'image des astronautes d'Apollo 11, nous voyons les robots effectuer de petits sauts d'un ou deux mètres.

2.3.5 Apprentissage « Lift Cube » de Widow

L'apprentissage de Widow, même pour une tâche simple telle que soulever un cube, possède déjà quelques degrés de complexité supplémentaires par rapport à l'apprentissage de Spider. Tout d'abord parce que le mouvement requis et l'interaction de contact sont beaucoup plus fins, et ensuite parce que cet apprentissage entre déjà dans un concept avancé : les curriculums. Dans cet apprentissage, l'objectif du robot est de s'approcher d'un cube, de le soulever avec son « end effector » et de le positionner à un point dans l'espace. On peut déjà imaginer que le robot doit réaliser successivement trois mouvements distincts pour réaliser sa mission.

2.3.5.1 Création de l'environnement

Les principes de création d'environnement dans Isaac Lab ont déjà été traités dans l'exemple Spider. Nous allons seulement décrire ici les observations, les actions et les conditions de reset utilisées pour l'apprentissage de Widow.

L'espace des actions est de dimension 8 car le robot possède 6 degrés de liberté, et le « gripper » est modélisé avec deux joints prismatiques. Le robot réel est fait avec un unique moteur pour le gripper. C'est pourquoi, pour s'affranchir de cette différence, il est possible d'ignorer la 8^{e} sortie du réseau de neurones et de prendre la 7^{e} (qui correspond donc à l'un des joints prismatiques du gripper), de l'inverser, et de s'en servir pour le 2^{e} joint du gripper.

Le robot est à base fixe. En conséquence, si le cube est éjecté trop loin pour que le bras puisse l'attraper, on considère que la mission n'est plus réalisable et on reset l'environnement. C'est la seule condition de reset que l'on utilise dans cet exemple.

Les observations du robot sont :

- La position et la vitesse de chaque articulation
- La position dans le référentiel de l'environnement du cube
- La distance entre l'end effector de Widow et le centre du cube
- Les actions de l'étape précédente

2.3.5.2 Calcul des Rewards

Les composantes de la fonction de reward de Widow sont :

- Reaching object reward : plus l'end effector est proche du cube, plus le robot gagne de points. Ce format de reward utilisant la tangente hyperbolique est très utilisé car il permet de définir une reward qui aura une influence seulement lorsque la valeur

mesurée (ici d) est dans la « zone de variation » de la fonction $tanh$. Dans la figure 27 on voit que la reward varie beaucoup lorsque d/std reste entre 0 et 2, puis au-delà de 2 ça ne varie plus beaucoup. Dans le cas de cette reward, si le robot est trop loin du cube il ne s'apercevra pas qu'il ne gagne pas de points. Mais dès qu'il s'approche, même par hasard, il va commencer à gagner des points et faire en sorte de continuer à exploiter cette reward. std (l'écart-type) permet de définir cette zone de variation de la fonction, c'est-à-dire la zone où la dérivée est grande.

$$d_{cube \to ee} = norm(pos_{cube} - pos_{ee})$$
$$Rew_{reaching\ object} = 1 - \tanh(\frac{d_{cube \to ee}}{std})$$

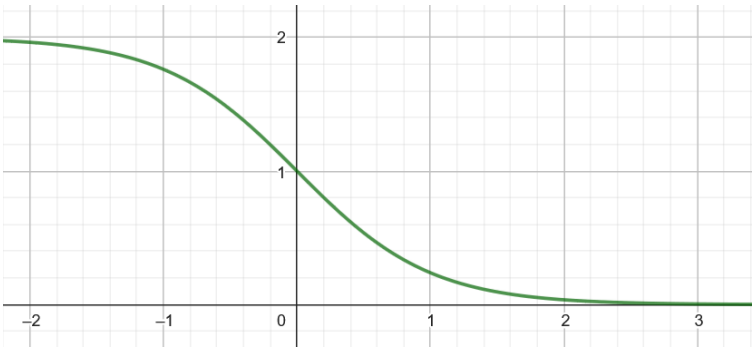


Figure 27 : Fonction 1 moins tangente hyperbolique

- Alignment reward : on récompense l'agent si son gripper reste aligné avec l'axe z du monde (donc s'il reste vertical)

$$Rew_{alignment} = abs(u_z \cdot \vec{z}_{ee})$$

- Lifting object reward : l'agent gagne des points s'il parvient à soulever le cube au-dessus d'un certain seuil

$$Rew_{lifting\ object} = \begin{cases} 1 & Si\ z_{cube} > minimal\ height \\ 0 & Sinon \end{cases}$$

- Object goal-tracking reward. Cette reward représente le 3e mouvement que le bras doit réaliser, c'est-à-dire positionner le cube à un endroit donné. En pratique, on utilise cette reward deux fois avec des std différents. Cela permet d'avoir une première approche grossière de l'objectif, puis lorsque le cube est presque à la bonne position, le robot commence à gagner encore plus de points, l'incitant ainsi à préciser le positionnement du cube.

$$d_{cube \to objectif} = norm(objectif - pos_{cube})$$
$$Rew_{object\ goal\ tracking} = 1 - \tanh(\frac{d_{cube \to objectif}}{std})$$

- Enfin, comme pour Spider, on donne des pénalités relatives à la quantité de mouvement des joints du robot.

2.3.5.3 Curriculums

En apprentissage par renforcement, un Curriculum est l'équivalent d'un programme d'entraînement. L'idée est qu'au lieu d'entraîner un agent directement sur une tâche très difficile, on commence par des versions plus simples et on augmente progressivement la difficulté. Par exemple, si on veut apprendre à un robot à marcher, on commence par l'entraîner sur un sol plat, ensuite on rajoute de petits obstacles, puis un terrain irrégulier… Quand on fait du Curriculum learning on peut donc modifier les paramètres physiques pendant l'apprentissage, mais on peut aussi changer les fonctions de rewards par exemple. Toutes les règles sont potentiellement modifiables. Lorsque l'agent fait face à un changement de règle, il voit sa performance globale baisser d'un coup. L'objectif pour lui est donc de s'adapter en se servant de ses expériences précédentes afin d'affiner sa politique.

Dans le cas de l'apprentissage « pick cube », on observe que le robot parvient à soulever le cube sans l'utilisation d'un curriculum. On a utilisé un gain pour les pénalités qui est très faible (10e-4). De cette manière le robot ne sera que très peu pénalisé par la quantité de ses propres mouvements, permettant à l'agent de ne pas être limité en mouvement. Ensuite le robot déplace le cube auprès de la position attendue, et tremble un peu pour garder cette position. C'est ici que le curriculum peut être utile dans cet exemple. Au bout de 1000 épisodes, on a choisi d'augmenter d'un facteur 1000 l'influence des pénalités. Le résultat est concluant : le robot se stabilise et reste presque immobile.

Bien que non obligatoire, l'utilisation de ce curriculum était avant tout un test pour comprendre comment fonctionne le concept et comment l'implémenter dans Isaac Lab qui fournit tous les outils nécessaires. Cela sera utile pour les futurs apprentissages du bras tels que l'apprentissage pour connecter une prise électrique. Dans cet apprentissage on utilisera aussi des curriculums pour changer la valeur des poids des rewards, mais aussi on utilisera un SBC (Sample-Based Curriculum) dont on parlera plus tard dans ce rapport.

2.3.5.4 Tuning des rewards

Pour ajuster les rewards du Widow afin qu'il attrape le cube, il a fallu procéder par petites étapes.

En premier lieu, le bras apprend à diriger son end effector vers son objectif, en l'occurrence le cube. La reward responsable de cela est la reaching reward. Très rapidement l'agent comprend comment converger vers le cube. Cependant il y va d'une manière qui n'est pas du tout naturelle. Il se recroqueville sur lui-même et racle le sol avec sa pince jusqu'à atteindre le cube. En l'atteignant il le pousse et l'éjecte dans une direction. Il essaie de le suivre mais le cube finit hors de sa portée. Pour améliorer ce mouvement on a essayé deux choses : d'abord un reset sur la distance entre le cube et la base du robot. Ensuite on a rajouté l'Alignment reward. Pour aider le bras on a aussi défini la position par défaut du robot de manière à ce que les mouvements pour atteindre le cube soient assez simples et pas trop amples.

Avec des gains respectivement de 1,5 et 1,0 pour les rewards reaching et alignment, le robot préfère favoriser son alignement avec l'axe Z. En conséquence il continue de se recroqueviller, tant que son gripper regarde vers le bas. En ajustant le gain d'alignement à 0.1, on obtient un bon premier résultat : le robot fond vers le cube, et il a même compris que pour gagner encore quelques centimètres sans éjecter le cube, il vaut mieux ouvrir la pince pour que le cube soit parfaitement à l'intérieur du gripper.

La grosse difficulté de cet apprentissage est de faire comprendre à l'agent que pour gagner encore beaucoup plus de points il doit fermer la pince et soulever le cube au-dessus d'un certain seuil (pas très haut pour favoriser les chances de réussite, mais pas trop bas non plus car l'agent pourrait avoir envie de faire sauter le cube sans chercher à l'attraper). C'est ici qu'intervient la reward lifting.

Pour « lifter » le cube, le robot doit déjà serrer la pince. Pour essayer de forcer l'agent à faire cela on a rajouté une reward relative à la distance entre chaque doigt de la pince et le centre du cube. On a mis un écart-type (std) très petit pour que cette reward ne commence à avoir une influence que lorsque le gripper est déjà proche du cube. Mais cela n'a pas fonctionné. Parfois il ne sert à rien de décrire des fonctions de récompense trop spécifiques, et on risque de finir par en ajouter beaucoup trop.

En essayant à tâtons plusieurs combinaisons de gains on a pu observer que le « switch » (la transition) à partir duquel le robot réussissait à soulever le cube ne se jouait à rien. Une combinaison pouvait fonctionner, puis un simple changement de paramètre ailleurs nous faisait revenir à un robot qui laisse le cube au sol. Par exemple après avoir réussi à combiner toutes les rewards décrites dans la partie précédente et à soulever le cube, on a modifié le mesh du gripper pour se rapprocher de ceux qu'on utilise sur le robot réel. Même si rien d'autre n'a changé, il n'y a plus le switch.

Une fois que le robot a appris à soulever le cube, il le place à un point de l'espace donné grâce aux rewards Object goal tracking. Dans nos premiers essais on observe que le bras apprend une position spécifique à cet objectif, mais qu'il ne généralise pas à d'autres objectifs. Donc quand on lui demande de positionner le cube à un autre endroit, il ignore cette commande et repositionne le cube au même endroit. C'est pourquoi on a voulu généraliser le plus possible l'apprentissage. Pendant l'entraînement, à chaque nouvel épisode on génère aléatoirement une position initiale du cube et des joints du bras, et on choisit un point objectif aléatoire aussi dans une boîte de l'espace. On observe que le switch met plus de temps à arriver mais on obtient à la fin un agent capable de prendre le cube et de le positionner à moins de 3cm de l'objectif tout en restant stable.

Voici ci-dessous deux séries de courbes de rewards (figures 28 et 29) correspondant à deux entraînements différents : un avec le switch, où l'agent comprend comment soulever le cube, et un autre sans.

La première série de courbes montre typiquement un apprentissage idéal. Les courbes de rewards lifting, goal gracking et reaching object restent très proches de 0 jusqu'à environ l'épisode 600 où les courbes ont une envolée puis finissent par stagner à partir de l'épisode 700. Reaching object monte jusqu'à 1,7 ce qui n'est pas très loin de son maximum (2) puis

diminue lentement jusqu'à 1,5. Au début l'agent accorde toute son importance à la distance au cube, puis quand il comprend que soulever le cube rapporte beaucoup plus de points, il délaisse légèrement cette récompense. On observe dans la simulation que le bras n'attrape le cube que du bout de la pince, et ce en un mouvement rapide et efficace, comme pour gagner du temps et donc des points.

Lifting et Object goal tracking stagnent presque à 15, leur valeur maximale, par contre la reward Object Goal tracking avec un écart type plus fin ne monte que jusqu'à 3 (par rapport à son maximum 5), indiquant que le robot ne parvient pas tout à fait à placer le robot exactement à la position souhaitée. En pratique on atteint une précision d'environ 3cm.

Enfin, les courbes de pénalités action rate et joint vel restent quasiment à 0 pendant tout le début de l'apprentissage, puis effectuent un saut jusqu'à respectivement – 7 et -0,7. C'est lié au curriculum qu'on a mis en place. Le point positif est que ces deux pénalités, qui interviennent lourdement sur l'agent au milieu de l'apprentissage, sont vraiment prises en compte car l'agent cherche à les faire remonter pour limiter la perte de points.

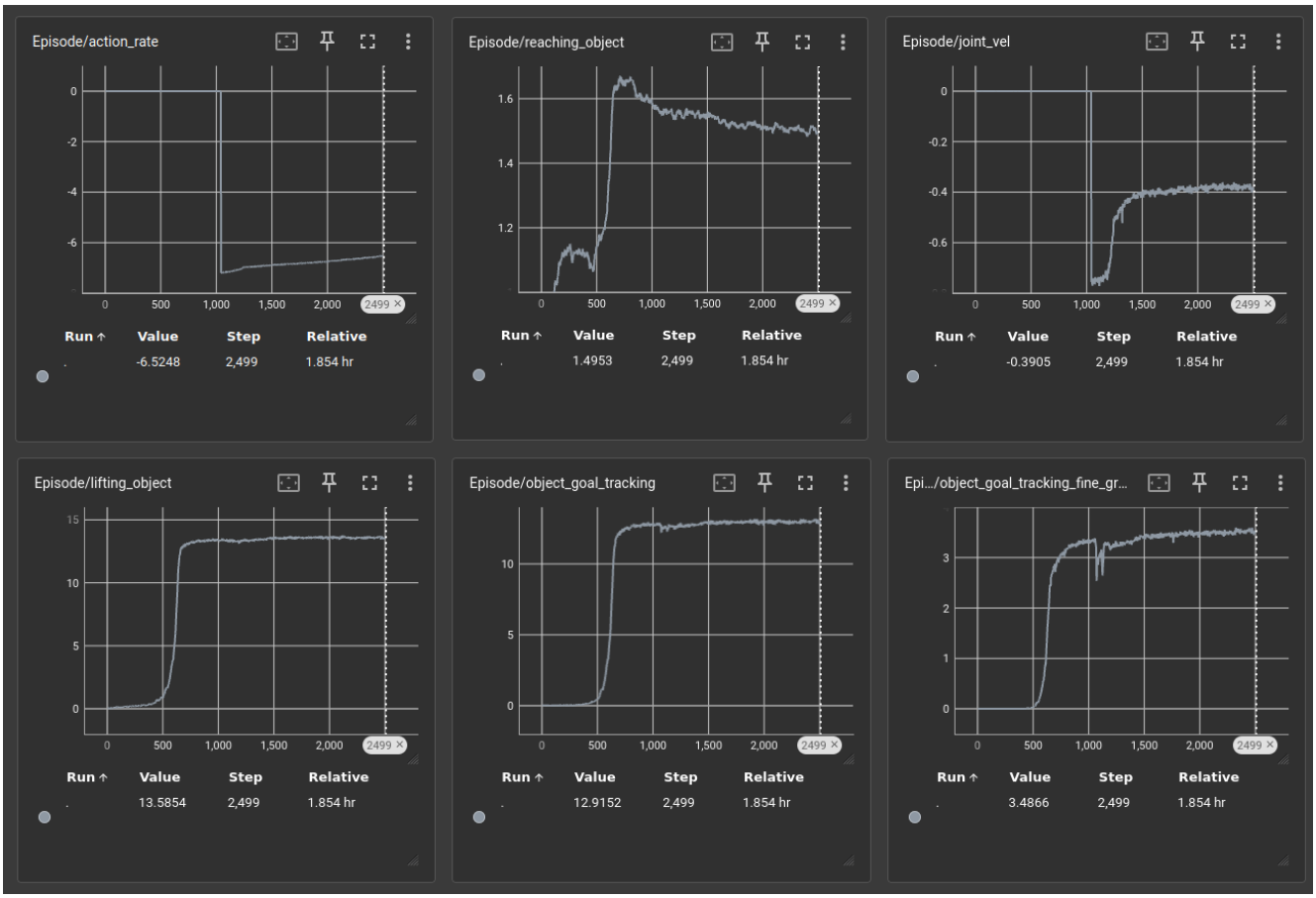


Figure 28 : Rewards pour un apprentissage avec "switch"

La deuxième série de courbes correspond au même entraînement, la différence était la combinaison de gains sur les différentes composantes de la reward. On observe que le switch ne s'effectue pas. Très vite le robot comprend comment se rapprocher du cube, la reaching reward arrive à 1,4 en à peine 200 étapes, mais les trois courbes de lifting, et object tracking ne décollent jamais. On a une bosse vers la 100e étape mais qui culmine à 0,025 pour lifting

par exemple. C'est lié au fait que certains agents, avant de penser à ouvrir la pince, tapent sur le cube et parviennent par chance à le faire décoller au-dessus du seuil de récompense.

Les courbes de pénalité montent naturellement, car l'agent met tout en œuvre pour gagner des points, en l'occurrence il limite ses mouvements.

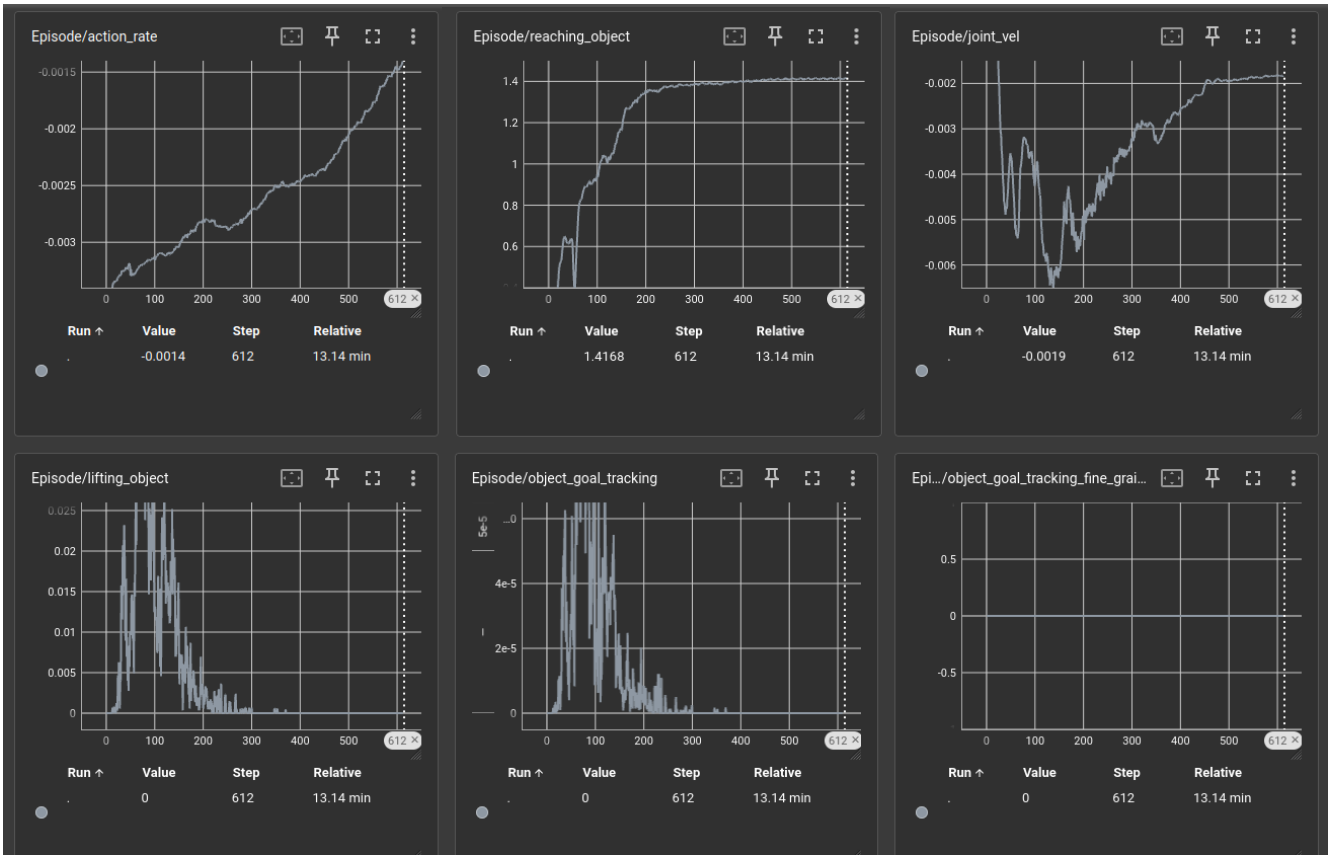


Figure 29 : Rewards sans "switch"

2.4 Transfert vers le réel

2.4.1 État de l'art

L'article « IndustReal : Transferring Contact-Rich Assembly Tasks from Simulation to Reality » [22] présente un ensemble de méthodes pour résoudre des tâches d'assemblage complexes en simulation en utilisant l'apprentissage par renforcement et pour transférer ces politiques apprises au monde réel. Ce projet s'appuie en partie sur les travaux du projet Factory [23] qui fournit des environnements d'apprentissage spécifiques aux manipulations d'assemblage en usine.

Cet article est particulièrement pertinent pour l'étude de la faisabilité du transfert de la simulation à la réalité dans des tâches robotisées. Le principal défi abordé est la difficulté d'effectuer des tâches d'assemblage nécessitant des interactions riches en contacts, comme le positionnement précis d'objets.

Traditionnellement, ces tâches sont réalisées par du matériel spécialisé, rendant les solutions coûteuses et peu adaptables. L'apprentissage en simulation offre une alternative pour former des agents robotisés à résoudre des tâches avant de les déployer dans des environnements réels.

Les techniques développées permettent aux robots de réaliser des tâches de type « Pick, Place and Insertion » en simulation et de transférer les politiques vers le monde réel. Pour les expérimentations, le robot utilisé est un Franka Emika Panda, équipé d'une caméra Intel RealSense 435 fixée à son poignet pour localiser les objets d'assemblage. Les objets manipulés comprennent plusieurs types d'assemblages industriels, tels que :

- Des tiges de différentes formes (rondes, rectangulaires) et tailles.
- Des engrenages de divers diamètres.
- Des connecteurs NEMA, correspondant à des prises américaines.

Les politiques d'apprentissage sont entraînées dans des environnements simulés sous Isaac Gym. Une fois l'entraînement terminé, ces politiques peuvent être déployées sur le robot réel sans ajustement supplémentaire. L'aspect le plus impressionnant réside dans la capacité de généralisation, car les politiques entraînées sur des piquets et des engrenages peuvent être appliquées pour brancher des prises.

Dans leurs travaux, l'espace d'observation inclut les angles des articulations du robot, les poses de la pince et de l'objet à manipuler, ainsi que les positions cibles pour chaque tâche. Les actions correspondent à des positions incrémentales à atteindre en utilisant le contrôleur, facilitant ainsi le contrôle en temps réel. Les récompenses sont variées et spécifiques à chaque tâche, reposant sur des critères tels que l'alignement, évalué grâce aux SDF, et une récompense terminale en cas de succès.

Le projet s'appuie sur quatre politiques distinctes : trois pour l'apprentissage en simulation et une pour le déploiement réel.

- Simulation-Aware Policy Update (SAPU) : cette méthode incite l'agent à éviter les erreurs de simulation, comme les interpénétrations entre objets. Elle pénalise les épisodes où les prédictions de simulation sont peu fiables, ce qui pousse l'agent à apprendre des comportements plus généralisables pour le transfert vers le monde réel. Le module vérifie les interpénétrations dans la simulation et ajuste les récompenses pour favoriser des politiques fiables

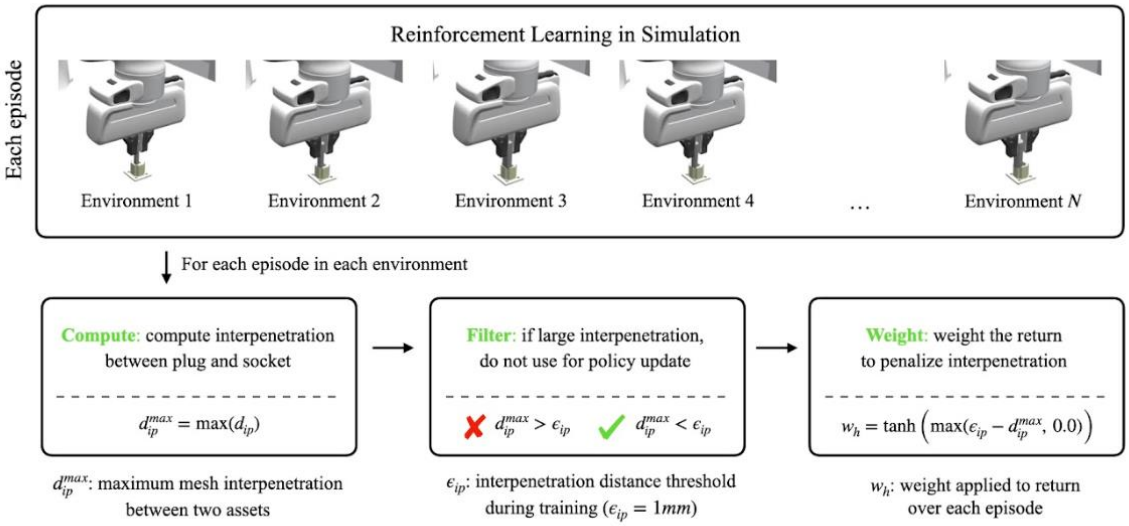


Figure 30 : Diagramme de fonctionnement de l'algorithme SAPU

- Signed Distance Field (SDF)-Based Reward : les récompenses sont générées en fonction de la distance signée entre les objets, permettant à l'agent de recevoir des signaux de récompenses continus en fonction de l'alignement des pièces d'assemblage. Cela améliore la précision de l'agent, notamment dans les tâches nécessitant un alignement très précis.

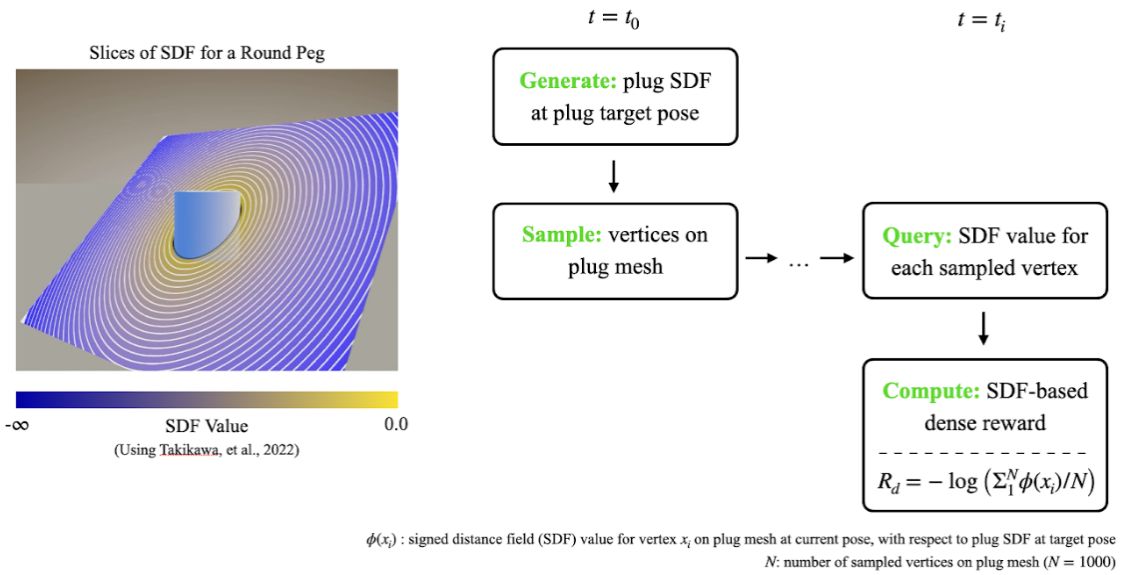


Figure 31 : Diagramme de fonctionnement de l'algorithme SDF-Based Reward

- Sampling-Based Curriculum (SBC) : on a déjà parlé du concept de Curriculum learning dans la partie précédente. Pour les tâches d'insertion, une variante nommée SBC permet à l'agent d'apprendre progressivement des scénarios plus complexes. L'agent commence par des états simples, la difficulté augmentant à mesure que la politique s'améliore, empêchant ainsi le surapprentissage et facilitant la généralisation

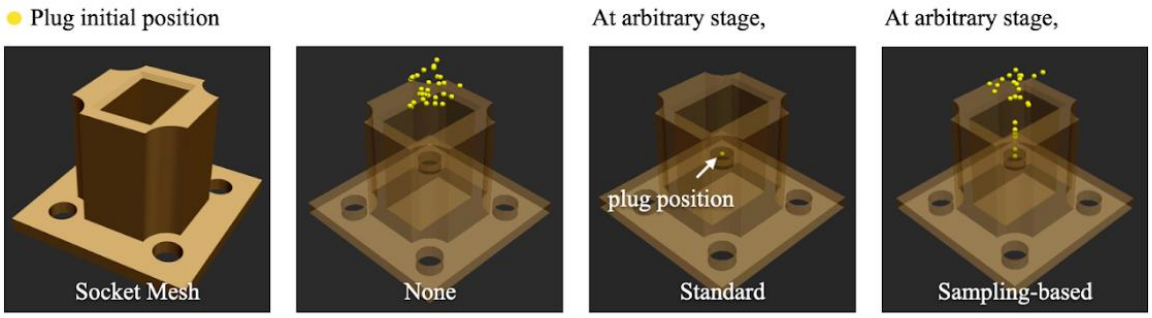


Figure 32 : Diagramme de fonctionnement de l'algorithme SBC

- Policy-Level Action Integrator (PLAI) : cette approche intègre les actions de la politique au fil du temps pour réduire l'erreur à l'état stationnaire lors du déploiement réel. Inspirée des régulateurs PID, elle améliore la précision en atténuant les erreurs causées par des perturbations non modélisées.

- We introduce Policy-Level Action Integrator (**PLAI**) as a policy deployment method. We define:
  - s_t: the current state,
  - s^d_{t+1}: the desired state,
  - a_t: be an action expressed as an incremental state target,
  - $\oplus$: computes the state update
- An established approach for applying policy actions is

$$s^d_{t+1} = s_t \oplus a_t$$

- In contrast, PLAI applies policy actions as

$$s^d_{t+1} = s^d_t \oplus a_t$$

- Unrolling from $t = 0...T$:

$$s^d_T = s_0 \oplus \sum_{i=0}^{T-1} a_i$$

Figure 33 : Fonctionnement de l'algorithme PLAI

Cet article démontre la faisabilité du transfert de politiques apprise en simulation vers des environnements réels pour des tâches d'assemblage complexes. En utilisant l'approche présentée dans cet article, il nous serait donc possible de réaliser les entraînements de branchements de prise et de vissage de connecteurs cryogéniques en simulation puis de tester cet apprentissage sur des interfaces réelles.

2.4.2 Travail préliminaire : la Proprioception robotique

La première étape avant de réaliser un transfert au réel, c'est de s'assurer que le robot n'est pas dangereux ni pour lui ni pour son environnement. En effet il n'est pas improbable qu'un bug se glisse dans le code et donne au robot des mouvements auxquels on ne s'attend pas. Dans les premières phases de la conception de Spider il est arrivé que le robot fasse un mouvement trop ample et brusque et casse une des pièces imprimées en 3D. Par chance c'est

l'un des seuls incidents qui a eu lieu mais on pourrait facilement imaginer Widow taper lourdement contre le sol avec sa pince par exemple, ou alors Spider se recroqueviller sur lui-même et créer des collisions internes. C'est dans cette optique qu'on a imaginé et implémenté quelques mesures de sécurité qu'on justifie dans ce paragraphe.

En premier lieu, on impose des limites physiques à la position des joints. Les moteurs Dynamixel rendent cela très facile, il suffit de régler les paramètres sur Dynamixel Wizard de *Max Position Limit* et *Min Position Limit*. Ces limites sont absolues, c'est-à-dire que lorsque l'utilisateur envoie une commande en position qui dépasse la limite, le moteur ignore la commande et reste immobile. A cela on a rajouté une surcouche logicielle telle que si la commande dépasse la limite, on envoie au moteur une autre commande qui correspond exactement à la limite. De cette manière le robot ne dépassera jamais ses limites mais il s'autorisera à se placer dessus.

Un tel paramétrage est non seulement utile car il réduit les possibilités de collisions internes des robots, mais en plus il favorise un apprentissage cohérent des agents. On a essayé plusieurs fois l'apprentissage de Spider avant de lui imposer des limites articulaires, mais le résultat n'a jamais été bon, le robot envoyant les pattes dans tous les sens, ou se cognant contre lui-même…

La deuxième mesure qu'on n'a pas encore implémenté à l'heure où j'écris ce rapport consiste à aider Widow à éviter les positions impossibles. Le cas le plus intuitif est celui où le robot cherche à placer son gripper en-dessous du niveau du sol, sans avoir conscience que le sol va le bloquer. L'idée pour éviter cela est de récupérer les commandes en position des servomoteurs avant de les envoyer dans les moteurs, et de faire des calculs de cinématique directe pour obtenir les positions de chaque membre du bras. Ensuite avec des équations de géométrie on peut savoir si les membres sont bien au-dessus du niveau du sol, sinon on bloque la commande.

La proprioception robotique pourrait constituer un sujet de PFE à part entière tant il est complexe de faire comprendre conscience à un robot de ses propres appendices. Dans la littérature on a trouvé quelques articles qui proposent des résultats très motivants, certains d'entre eux utilisant même l'apprentissage par renforcement. [24], [25]

2.4.3 Transfert progressif

Le nerf sensible du transfert réside dans toutes les petites différences qui existent entre la simulation et le robot réel. C'est pourquoi il est important de maîtriser les paramètres physiques tels que les coefficients de friction dynamique/statique, ou encore les paramètres de contrôle des servomoteurs, que l'on a étudié dans la partie 2.1. Cette section traite de tous les travaux effectués dans le but de réaliser un transfert en boucle fermée qui soit satisfaisant.

2.4.3.1 Tests en boucle ouverte

Un test en boucle ouverte consiste à actionner le robot réel à partir de commandes générées par la simulation. On ne met pas en jeu les observations réelles de l'environnement, mais on prend celles de la simulation pour générer *via* le réseau de neurones entraîné le vecteur d'action. Ce test comporte plusieurs avantages. Pour commencer on s'assure que le

comportement qu'on observe sur l'écran correspond bien à celui du vrai robot. Tout le travail qui consiste à mapper les commandes de la simulation vers les commandes des moteurs réels restera utile lorsqu'on passera en boucle fermée car dans tous les cas le réseau de neurones donne le même type de commandes.

Par ailleurs la boucle ouverte permet aussi de contrôler la vitesse d'exécution de la mission. Les étapes de la simulation et de l'apprentissage se déroulent à intervalles de temps 1/60s, c'est-à-dire 60 fois par seconde. Et à chaque étape une nouvelle commande est générée par l'agent. On a voulu dans un premier temps ralentir ce processus en créant un script qui joue les étapes une par une à chaque pression de la touche entrer. Pour Spider ce test n'a pas été concluant car l'agent a besoin d'effectuer les mouvements dans des timings précis pour se propulser tout en gardant son équilibre. En revanche Widow n'est pas gêné par ce test car même en prenant son temps pour s'approcher du cube, ce dernier restera immobile à sa place. Il n'y a donc aucune contrainte de temps.

On adapte au besoin une fonction pour mapper les valeurs de la simulation à la valeur réelle qu'il faut envoyer au servomoteur. Par exemple pour Widow le gripper réel et celui de la simulation ne sont pas modélisés de la même manière mécaniquement. Il faut donc transformer une action donnée en m et la convertir en *tic* moteur (qui correspond donc à une valeur d'angle) :

$$y_{tic} = -\frac{x_m}{0.03} * (2669 - 1533) + 1533$$

2.4.3.2 Corrélation réel/simulation

A ce stade nous avons passé beaucoup de temps à affiner la similarité entre simulation et réel. Ce travail peut être fastidieux mais il est nécessaire pour optimiser l'apprentissage et le bon déroulé du transfert. Sauf dans certains cas, il est souvent plus facile d'adapter la simulation au robot réel que l'inverse.

Une des modifications importantes faites à Spider a été de réduire la longueur de ses pattes. Cela a été décidé lorsqu'on a vu que les servomoteurs du robot renvoyaient régulièrement des *Overload Error*. Ce sont des messages qui interviennent lorsqu'un moteur dépasse la charge maximale en couple qu'il est autorisé à encaisser. Ici les moteurs sont limités à 1,5 Nm. En réalité ils peuvent probablement aller au-delà de cette limite mais par sécurité le constructeur a placé ce failsafe pour prévenir tout problème de surchauffe. C'est un paramètre auquel nous n'avons pas fait attention pendant l'apprentissage, ce qui a permis à certains agents de développer des méthodes de déplacement par petits sauts par exemple.

Nous avons donc réduit la longueur de la première section de chaque patte :

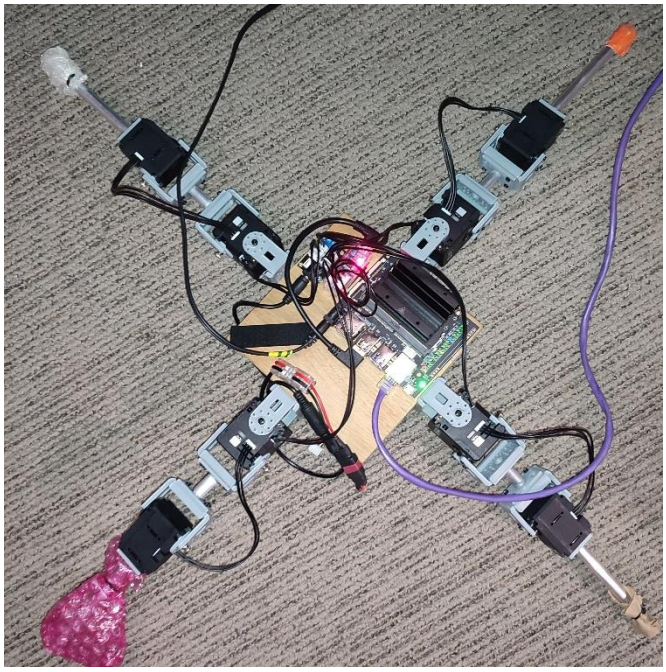


Figure 34 : Modèle final de Spider

Réduire la longueur des pattes réduit l'effort de bras de levier sur les deux articulations du plan horizontal. Pour essayer de mieux comprendre les efforts présents dans le système nous avons comparé, pour une position statique donnée (la position « debout ») les couples de chaque joint dans la simulation et dans la plateforme réelle. A l'heure actuelle on observe beaucoup de décalage entre les deux. Un objectif dans le prochain mois sera de comprendre pourquoi et d'essayer de corriger cela. En revanche nous avons pu observer, en mettant à l'épreuve manuellement la force des moteurs, que ceux-ci sont relativement robustes et ne devraient pas atteindre l'overload aussi fréquemment, au vu du poids du robot. Il y a donc sûrement des paramètres de damping et stiffness à régler.

Pour rappel, ces paramètres interviennent dans la formule du contrôle PD :

$$force = stiffness\,.\,ecart_{pos} + damping\,.\,ecart_{vitesse}$$

Ces deux paramètres nous ont aussi tendu un autre piège : lorsqu'on calcule un vecteur action et qu'on l'envoie dans les moteurs, stiffness et damping contrôlent la vitesse à laquelle le servomoteur va converger vers sa commande. Et il peut arriver qu'entre deux étapes de la simulation, les articulations n'aient pas le temps de converger. Puis une nouvelle action est envoyée et coupe la route des moteurs qui n'arrivent donc jamais à la position souhaitée à l'étape précédente.

Enfin, pour faire une estimation de la réelle valeur des coefficients de friction dynamique et statique entre les pattes du robot et le sol sur lequel il évolue, nous avons réalisé des tests empiriques. Nous avons essayé les algorithmes de mouvement manuel pour voir comment se comportait le robot sur différentes surfaces (sur de la moquette très adhérente, sur du métal lisse et sur du papier), et pour chaque surface nous avons cherché les paramètres dans la simulation qui nous amenaient vers un résultat le plus similaire possible. La moquette nous pose problème car elle est tellement adhérente qu'elle s'accroche aux pattes du robot qui sont en métal découpé à la scie, c'est-à-dire pas très lisse. Nous avons donc essayé de rajouter des

chaussettes à Spider pour essayer de diminuer la valeur du coefficient et pour que les pattes ne s'accrochent plus. Il a été difficile de trouver un coefficient réaliste dans la simulation, cela doit être dû au fait que la moquette est une surface un peu molle, ce qui rend une modélisation du sol comme surface solide invalide. Le papier est légèrement glissant mais on a pu modéliser un coefficient un peu en-dessous de 1,0, avec lequel le robot patine un peu mais arrive à avancer. Sur le métal en revanche, le robot réel glisse complètement et ne parvient que très peu à avancer. On a alors mis un coefficient proche de 0 dans la simulation.

2.4.4 Transfert en boucle fermée

La boucle fermée est un test de transfert de l'apprentissage vers la plateforme réelle dans lequel le robot est en totale autonomie. Il va utiliser ses propres observations (réelles) pour calculer ses actions.

Après tout le travail préliminaire réalisé, tout ce qu'il reste à faire pour réaliser un test en boucle fermée est de réunir en un vecteur toutes les observations nécessaires à l'agent. C'est le vecteur d'input du réseau de neurones. L'ensemble du stage n'était pas axé sur les capteurs ou la vision du robot sur son environnement. Nous n'avons donc pas travaillé sur l'intégration de caméra ou de Lidar, qui auraient permis de construire proprement ce vecteur d'observations. A la place nous avons réalisé des approximations

Pour Widow, on fournit une position du cube plus ou moins proche de la réalité jusqu'à ce que le bras arrive à l'attraper. Ensuite on considère que le cube est au niveau du end effector jusqu'à la fin. Dès que le bras attrape le cube, le cube est censé être en mouvement et donc sa position change. Pour que le robot accède lui-même à cette position il faudrait utiliser par exemple une caméra 3D. Cette seule approximation est suffisante pour générer un vecteur d'observations car le reste du vecteur est facile à obtenir avec des données réelles qui proviennent par exemple des servomoteurs.

Pour Spider c'est plus délicat car nous avons besoin de fournir au vecteur d'observation la position du torse du robot. Normalement il nous faudrait par exemple une IMU avec GPS intégré pour qu'un filtre de Kalman nous fournisse une prédiction de la position. Sans cette estimation nous avons jugé acceptable de considérer que le robot resterait virtuellement à l'origine de l'environnement. Car normalement l'agent devrait avoir appris à se déplacer même à la position 0.

La sortie du réseau de neurones peut être très imprévisible. C'est pourquoi il faut réaliser un filtrage fréquentiel des commandes. Cela consiste à faire passer dans un filtre passe-bas les actions permettant ainsi de s'affranchir de toutes les commandes qui feraient « trembler » le robot.

Enfin, la dernière chose à prendre en compte est la fréquence d'actualisation de l'action. La simulation tourne à 60Hz. Il faut donc faire tourner la boucle fermée à 60Hz aussi pour que le robot ne soit pas déboussolé par rapport à la simulation.

A l'heure de l'écriture de ce rapport, je n'ai pas encore eu le temps d'effectuer de test en boucle fermée. Les scripts sont presque prêts et les démarches décrites dans cette partie sont

presque toutes implémentées. On compte sur le mois de stage qui reste pour effectuer ces tests.

3 Conclusion

3.1 Résultats

3.1.1 Validité des résultats par rapport au problème

Les projets de démonstrateurs de lanceurs réutilisables du CNES répondent à des défis majeurs, notamment pour la préparation, la récupération et la sécurisation des lanceurs. Le premier objectif de mon stage consistait à étudier un simulateur physique permettant de reproduire avec précision les conditions réelles d'opérations. Ce simulateur, avec la modélisation détaillée des interactions physiques de contact et des efforts se rapproche de la réalité et permet un paramétrage très fin des modèles simulés de nos robots. Dans le simulateur, des tâches de manipulation telles que « Pick Cube » ou encore la locomotion de Spider ont été réalisées avec succès via l'utilisation de méthodes classiques, telles que la cinématique inverse pour Widow.

L'étude des méthodes de machine learning adaptées à la robotique s'est montrée efficace et intéressante. C'est un travail précurseur vers des apprentissages beaucoup plus complexes qui amèneront à terme à la réalisation de tâches de passivation des démonstrateurs de lanceurs réutilisables. Les résultats obtenus en simulation démontrent qu'il est possible d'exécuter des manipulations robotiques pour des tâches complexes. Ces résultats laissent espérer une adaptation efficace des modèles au problème spécifique des tâches d'assemblage, afin d'atteindre des performances satisfaisantes dans les actions d'insertion et de vissage.

Le second objectif visait à évaluer la faisabilité du transfert d'apprentissage de la simulation vers des applications en conditions réelles. Encore en cours d'étude, les premières manipulations sont motivantes et laissent espérer de bonnes réalisations de leurs missions dans le futur pour les robots Widow et Spider.

3.2 Perspectives d'amélioration

Pour prolonger le travail débuté lors de ce stage, plusieurs axes d'améliorations et développements futurs pourront être envisagés, afin de valider complètement les résultats et d'optimiser le projet pour les besoins opérationnels du CNES.

D'un point de vue robotique, des modifications sur Spider pourraient participer à le rendre plus léger, plus mobile et robuste. Ses apprentissages pourraient inclure des missions bien plus évoluées que simplement marcher tout droit ; on pourrait par exemple imaginer lui apprendre à sortir d'un labyrinthe. Cela nécessite de lui intégrer des moyens de perception, comme une caméra 3D ou *a minima* un Lidar. Widow est déjà un robot robuste, mais sans doute un peu trop petit pour participer aux missions réelles sur des lanceurs réutilisables. Il ne sert que de démonstrateur, c'est pourquoi la continuité de ce projet pourrait inclure des tests sur les bras robotiques utilisés à l'heure actuelle.

Au niveau des apprentissages, il serait judicieux d'intégrer les modèles proposés dans les projets Factory et IndustReal sous Isaac Lab. La poursuite du projet pourrait inclure l'intégration de ces modèles pour développer un apprentissage avancé, qui pourrait ensuite être transféré dans le réel pour vérifier son efficacité. Si les résultats sont concluants, les mêmes algorithmes pourront être appliqués à des tâches d'assemblage plus complexes tels que les mouvements de reconnexion électriques et fluidiques. Dans un premier temps des tests avec Widow et des maquettes de connecteurs réalisées à l'aide de l'imprimante 3D pourraient être réalisés, puis dans un second temps, l'utilisation de connecteurs plus proches de ceux utilisés dans le domaine spatial pourrait être envisagé. A plus long terme, la modélisation des interfaces réelles des lanceurs dans le simulateur permettrait d'étudier et de tester des scénarios précis d'assemblage et de manipulation. Les phases d'apprentissage réalisées en simulation pourront être transférées pour tester directement les solutions sur les démonstrateurs réels. En cas de réussite, cela offrirait une alternative viable aux robots actuellement prévus ou du moins à leur programmation, en développant une solution robotisée plus autonome.

Enfin, dans l'optique de se ramener à des scénarios réalistes, on pourrait mettre à jour le simulateur pour intégrer aux scénarios les aléas d'un atterrissage comme la fumée, les vibrations, les chocs, autant de capteurs qui peuvent interférer avec le robot, sa robustesse et sa perception de l'environnement.

3.3 Conclusion

3.3.1 Conclusion du projet

Ce stage s'inscrit dans une démarche d'innovation et de transformation pour le CNES, répondant aux enjeux actuels de réutilisabilité des lanceurs spatiaux. Le travail accompli a permis de poser des bases solides pour des solutions robotiques autonomes capables d'effectuer des opérations critiques de préparation, récupération et sécurisation des démonstrateurs. L'étude d'un simulateur physique a ouvert la voie à des tests approfondis en environnement contrôlé, permettant de valider les premières étapes de la manipulation robotiques sans les coûts et les contraintes des essais physiques. En plus de sa capacité à simuler des interactions physiques réalistes, le simulateur offre également un potentiel de personnalisation et de réutilisation pour d'autres projets robotiques au CNES.

Le cadre mis en place durant ce projet a permis de valider des processus de conception de plateformes autonomes, de la modélisation en simulation au transfert de politiques, en passant par la conception « from scratch » de plateformes robotiques. Ce cadre technique, à la fois robuste et adaptable, constitue une base prometteuse pour l'évolution du projet.

Les résultats sont encourageants mais soulignent également les défis qui restent à relever. Les premières expérimentations en simulation, puis en conditions réelles, ont validé des cas de manipulations robotiques de base. Cependant, des travaux supplémentaires sont nécessaires pour atteindre une robustesse et une précision suffisantes dans des scénarios de plus grande complexité, comme les insertions de connecteurs complexes pendant des opérations de lancement ou post-atterrissage. L'exploration du transfert d'apprentissage de la simulation

vers le réel représente une avancée potentielle pour surmonter ces défis et améliorer la capacité des robots à exécuter des missions complexes de manière autonome.

3.3.2 Conclusion personnelle

Ce stage au CNES a représenté une étape clé dans mon parcours, marquant la fin de mes études d'ingénieur et clarifiant les orientations professionnelles que je souhaite suivre. À travers ce projet, j'ai eu l'opportunité de consolider et d'approfondir mes compétences en mobilisant un large éventail de connaissances, allant de l'électronique aux mathématiques appliquées à l'apprentissage par renforcement.

Cette approche multidisciplinaire m'a permis de travailler sur un pipeline de conception d'agent robotique autonome, en prêtant attention à chaque détail technique et en assurant une intégration cohérente des différentes briques fonctionnelles du projet. J'en retire une vision d'ensemble précieuse, ainsi qu'une meilleure compréhension des enjeux liés à des systèmes complexes et ambitieux.

La diversité et la richesse des missions qui m'ont été confiées ont renforcé mon intérêt pour les domaines de la robotique et de l'intelligence artificielle, une combinaison particulièrement stimulante. J'y ai confirmé un goût pour les défis intellectuels et de la résolution de problématiques complexes en lien avec la robotique. Ce goût pour l'innovation s'est affirmé au fil du projet, et j'ai aujourd'hui la volonté de poursuivre dans des environnements où l'expérimentation, la recherche et l'apprentissage continu sont au cœur de l'activité.

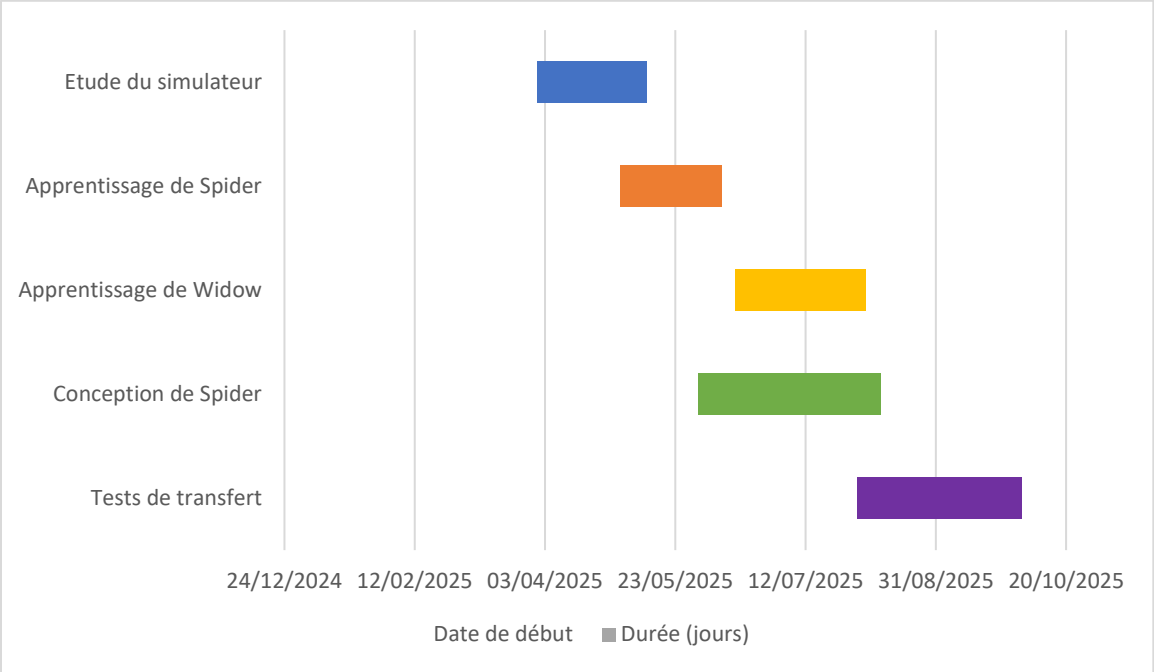
Une certaine frustration subsiste néanmoins : celle de ne pas avoir pu mener le projet à terme, notamment en ce qui concerne l'aspect du transfert d'apprentissage. A l'heure actuelle il me reste un mois de stage que je compte mettre à profit pour approfondir les apprentissages et les manipulations robotiques dans l'optique de réaliser des transferts robustes et satisfaisants.

Cette expérience m'a également permis de me former à Isaac Sim, un outil qui s'annonce incontournable dans le domaine de la robotique. Cette compétence enrichit mon profil et m'ouvre de nouvelles perspectives, notamment au regard des avancées actuelles de NVIDIA sur le transfert de politiques, que je compte suivre de près.

4 Annexes

4.1 Table des figures

4.2 Diagramme de Gantt



4.3 Figures supplémentaires

Figure 35 : Franka Emika Panda Robot

Figure 36 : Wlkata

Figure 37 : Spider sur différentes surfaces. Moquette, Métal, Papier

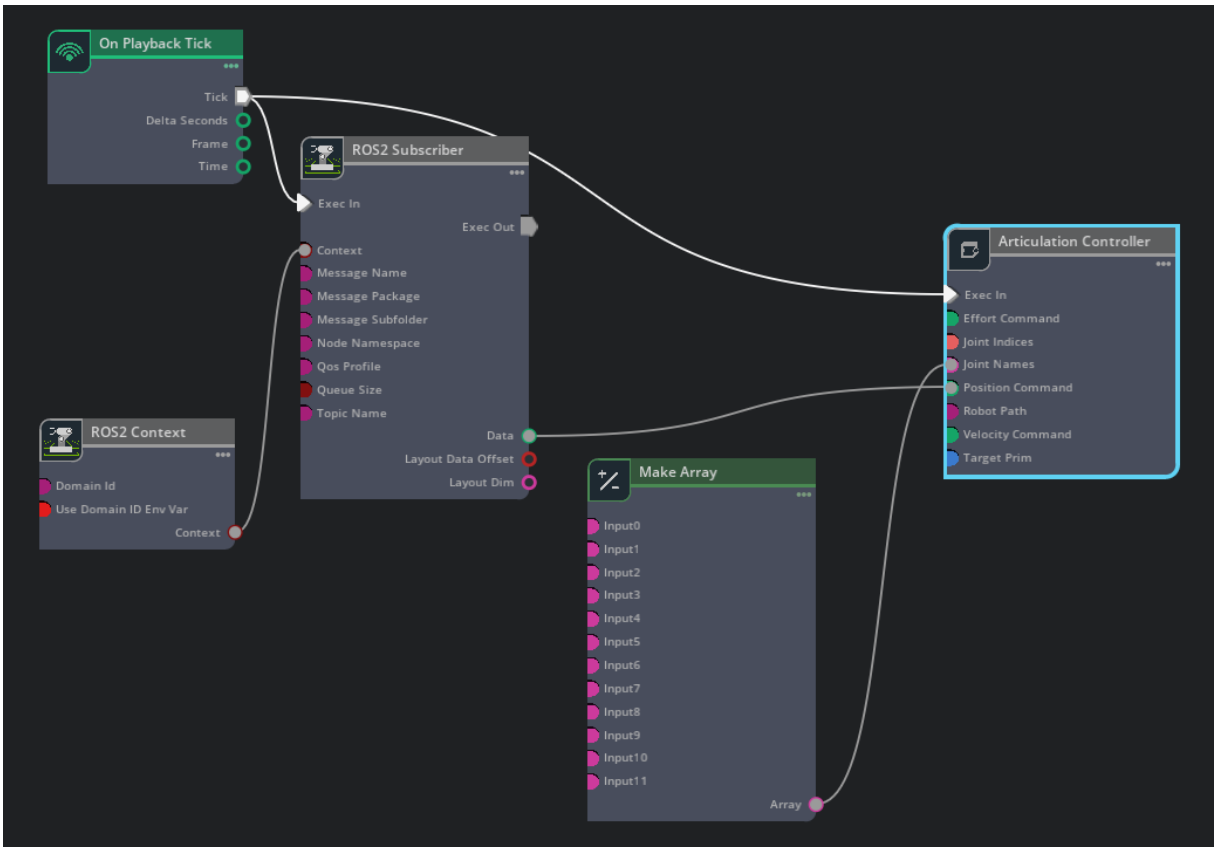


Figure 38 : Action Graph permettant de lire un topic ROS pour partager les commandes réelles à la simulation