gz-unitree: Reinforcement learning en robotique avec validation par moteurs de physique multiples pour le robot H1v2 d'Unitree

Gwenn Le Bihan
gwenn.lebihan7@gmail.com
ENSEEIHT

3 Novembre 2025

1 Remerciements

Je tiens à remercier Olivier Stasse et Guilhem Saurel, qui m'ont suivie pendant toute la durée du stage et répondu à mes questionnements, sans qui je n'aurais pu mener ces recherches. Merci aussi à Côme Perrot, grâce à qui j'ai pu éclaircir certaines zones d'ombre sur ma compréhension de la théorie du reinforcement learning appliquée à la robotique.

Merci à Sylvie Chambon, Géraldine Morin et Ronan Guivarch, professeurs à l'ENSEEIHT, qui m'ont soutenue à travers des périodes parfois difficiles.

Merci aussi à Laurenz Mädje et Marin Haug pour avoir créé Typst, une alternative moderne à LaTeX qui a rendu l'écriture de ce rapport bien plus agréable. Merci à Joseph Wilson (paquet Typst « Fletcher ») et Johannes Wolf (paquet Typst « CeTZ »), qui ont rendu la création de diagrammes très ergonomique.

Table des matières

[1] Domain-Specific Language

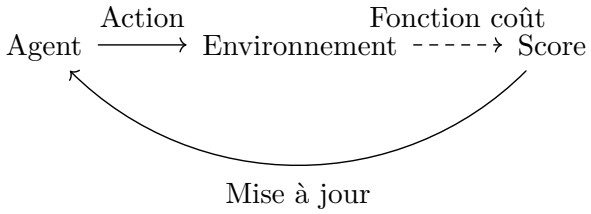
2 Contexte

2.1 Bases théoriques du *Reinforcement Learning*

L'apprentissage par renforcement, ou *Reinforcement Learning*, permet de développer des programmes sans expliciter leur logique: on décrit plutôt quatre choses, qui vont permettre à la logique d'émerger pendant la phase d'entraînement:

- Un *agent*: c'est le programme que l'on souhaite créer
- Des *actions* que l'agent peut choisir d'effectuer ou pas
- Un *environnement*, que les actions viennent modifier
- Un *score* (*coût* s'il doit être minimisé, *récompense* inversement) qui dépend de l'état pré- et post-action de l'environnement ainsi que de l'action qui a été effectuée

La phase d'apprentissage consiste à trouver, par des cycles d'essai/erreur, quelles sont les meilleures actions à prendre en fonction de l'environnement actuel, avec meilleur définit comme « qui minimise le coût » (ou maximise la récompense):



Cette technique est particulièrement adaptée au problèmes qui se prêtent à une modélisation type « jeu vidéo », dans le sens où l'agent représente le personnage-joueur, et le coût un certain score, qui est condition de victoire ou défaite.

En robotique, une approche similaire explore l'espace d'action (en général un courant à envoyer aux moteurs) de façon à optimiser le coût.

En robotique, on a des correspondances claires pour ces quatres notions:

Agent Robot pour lequel on développe le programme de contrôle (appelée une *politique*)

Actions Envoi d'ordres aux moteurs, souvent le courant électrique à appliquer

Environnement Le monde réel. C'est de loin la partie la plus difficile à simuler informatiquement. On utilise des moteurs de simulation physique, dont la multiplicité des implémentations est importante, voir Chapitre 2.1.3.3

Coût un ensemble de contraintes (« ne pas endommager le robot ») et d'évaluations spécifiques à la tâche à effectuer (« s'est déplacé de 5m en avant selon l'axe x).

2.1.1 L'entraînement

Une fois que ce cadre est posé, il reste à savoir *comment* l'on va trouver la fonction qui associe un état de l'environnement à une action.

Une première approche naïve, mais suffisante dans certains cas, consiste à faire une recherche exhaustive et à stocker dans un simple tableau la meilleure action à faire en fonction d'un état de l'environnement:

État actuel (x, retour)	**Meilleure action** $+1$ ou -1	**Coûts associés**
(0, C'est plus)		
(1, C'est plus)		
(3, C'est moins)		
(4, C'est moins)		
(5, C'est moins)		

Tableau 1. – Mémoire exhaustive initiale pour un « C'est plus ou c'est moins » dans $\{0, 1, 2\}$, avec pour solution 2

L'entraînement consiste donc ici en l'exploration de l'entièreté des états possibles de l'environnement, et, pour chaque état, le calcul du coût associé à chaque action possible.

Il faut définir la fonction de coût, souvent appelée L pour *loss*:

$$L : E \to S \tag{1}$$

avec E l'ensemble des états possibles de l'environnement, et S un ensemble muni d'un ordre total (on utilise souvent $[0, 1]$). Ces fonctions coût, qui ne dépendent que de l'état actuel de l'environnement, représente un domaine du RL[2] appelé *Q-Learning* [1]

On remplit la colonne « Action à effectuer » avec l'action au coût le plus bas:

État actuel (x, retour)	**Meilleure action** $+1$ ou -1	**Coûts associés** avec $L = (x, \text{retour}) \mapsto \lvert x - 2 \rvert$
$(0, \text{C'est plus})$	$+1$	$L(x+1,) = 2 \quad L(x-1,) = 2$
$(1, \text{C'est plus})$	$+1$	$L(x+1,) = 1 \quad L(x-1,) = 2$
$(3, \text{C'est moins})$	-1	$L(x+1,) = 2 \quad L(x-1,) = 3$
$(4, \text{C'est moins})$	-1	$L(x+1,) = 3 \quad L(x-1,) = 4$
$(5, \text{C'est moins})$	-1	$L(x+1,) = 4 \quad L(x-1,) = 5$

Tableau 2. – Entraînement terminé, avec pour fonction coût L la distance à la solution

Ici, cette approche exhaustive suffit parce que l'ensemble des états possibles de l'environnement, E, posssède 6 éléments

Cependant, ces ensembles sont bien souvent prohibitivement grands (e.g. $x \in [\![0, 10^{34}]\!]$), infinis ($x \in \mathbb{N}$) ou indénombrables ($x \in \mathbb{R}$)

Dans le cas de la robotique, E est une certaine représentation numérique du monde réel autour du robot, on imagine donc bien qu'il y a beaucoup trop d'états possibles.

2.1.2 Deep Reinforcement Learning

Une façon de remédier à ce problème de dimensions est de remplacer le tableau exhaustif par un réseau de neurones:

État actuel	devient la couche d'entrée
Meilleure action	devient la couche de sortie
Coûts associés	la fonction à optimiser par le réseau. Il peut s'agir d'une fonction qui permet au réseau de neurones d'approximer une autre fonction par supervision.
Le remplissage du tableau	devient la rétropropagation pendant l'entraînement

[2] Reinforcement Learning

Mise à jour (*Q-learning*)

Le score associé à un état s_t et une action a_t, appelée $Q(s_t, a_t)$ ici pour « quality » [2] ou « action-value » [3], est mis à jour avec cette valeur [4]:

$$(1-\alpha)\underbrace{Q(s_t,a_t)}_{\text{valeur actuelle}} + \alpha\left(\underbrace{R_{t+1}}_{\substack{\text{récompense}\\ \text{pour cette action}}} + \gamma \underbrace{\max_a Q(S_{t+1},a)}_{\substack{\text{récompense de la meilleure}\\ \text{action pour l'état suivant}}}\right) \tag{2}$$

L'expression comporte deux hyperparamètres, à valeurs dans $]0,1[$:

Learning rate α contrôle à quel point l'on favorise l'évolution de Q ou pas.

Discount factor γ contrôle l'importance que l'on donne aux récompenses futures. Il est utile de commencer avec une valeur faible puis l'augmenter avec le temps [5].

2.1.3 Difficultés liées à l'implémentation de la fonction coût

Tendances à la « tricherie » des agents

Expérimentalement, on sait que des tendances « tricheuses » émergent facilement pendant l'entraînement [Réf. nécéssaire]: l'agent découvre des séries d'actions qui causent un bug avantageux vis à vis du coût associé, soit parce qu'il y a un bug dans le calcul de l'état de l'environnement post-action, soit parce que la fonction coût ne prend pas suffisemment bien en compte toutes les possibilités de l'environnement (autrement dit, il manque de contraintes).

Dans le cas de la robotique, cela arrive particulièrement souvent, et il faut donc un simulateur qui soit suffisamment réaliste.

Sous-spécification de la fonction coût

(Note: Bof cette partie)

Un exemple populaire est l'expérience de pensée du Maximiseur de trombones [6]: un agent avec pour environnement le monde réel, pour actions « prendre des décisions »; « envoyer des emails »; etc. et pour fonction récompense (une fonction à maximiser au lieu de minimiser) « le nombre de trombones existant sur Terre », finirait possiblement par réduire en escalavage tout être vivant capable de produire des trombones: la fonction coût est sous-spécifiée

La validation comme méthode de mitigation

Comme ces bugs sont des comportements non voulus, il est très probables qu'ils ne soient pas exactement les mêmes d'implémentation à implémentation du même environnement.

Il convient donc de se servir de *plusieurs* implémentations: un sert à la phase d'entraînement, pendant laquelle l'agent développe des « tendances à la tricherie », puis une phase de *validation*.

Cette phase consiste en le lancement de l'agent dans une autre implémentation, avec les mêmes actions mais qui, crucialement, ne comporte pas les mêmes bugs que l'environnement ayant servi à la phase d'apprentissage.

Les « techniques de triche » ainsi apprises deviennent inefficace, et si le score (le coût ou la récompense) devient bien pire que pendant l'apprentissage, on peut détecter les cas de triche.

On peut même aller plus loin, et multiplier les phases de validation avec des implémentations supplémentaires, ce qui réduit encore la probabilité qu'une technique de triche se glisse dans l'agent final

(Note: Rien à voir mais je me dis, c'est enfait un moyen de trouver des bugs dans un physics engine ! ça me fait penser au Fuzzing un peu, mais avec un NN plutôt que du hasard contrôlé)

2.2 Évaluation de la performance d'une politique

Théoriquement, le « score » associé à un couple état/action est souvent réduit à l'intervalle $[0, 1]$ et assimilé à une distribution de probabilité: Q est une fonction de $S \times A$ vers $[0, 1]$ qui renvoie la probabilité qu'a l'agent à choisir une action en étant dans un état de l'environnement.

On note dans le reste de cette section:

$\boldsymbol{A}$	l'ensemble des actions
$\boldsymbol{S}$	l'ensemble des états possibles de l'environnement
$\boldsymbol{\rho_0 : S \to [0, 1]}$	la distribution de probabilité de l'état initial de l'environnement. Si l'on initialise l'environnement de manière uniformément aléatoire, ρ_0 est une équiprobabilité[3]
$\boldsymbol{M : S \times A \to S}$	le moteur de simulation physique, qui applique l'action à un état de l'environnement et envoie le nouvel état de l'environnement
$\boldsymbol{\Pi : S \to A}$	une politique
$\boldsymbol{\Pi^* : S \to A}$	la meilleure politique possible, celle que l'on cherche à approcher
$\boldsymbol{R : S \to \mathbb{R}^+}$	sa fonction de récompense
$\boldsymbol{Q_\pi : S \times A \to [0, 1]}$	sa distribution de probabilité, qu'on suppose Markovienne (elle ne dépend que de l'état dans lequel on est). $Q_\pi(s_t, a_t)$ est la probabilité que π choisisse a_t *quand on est dans l'état* s_t (s_t est l'état **pré**-action, et non post-action)
$\boldsymbol{Q}$ **et** $\boldsymbol{Q^*}$	Q_Π et Q_{Π^*}, pour alléger les notations

On suppose A et S dénombrables[4].

Pour alléger les notations, on surchargera les fonctions récompenses pour qu'elle puissent prendre en entrée des éléments de $S \times A$, en ignorant simplement l'action choisie:

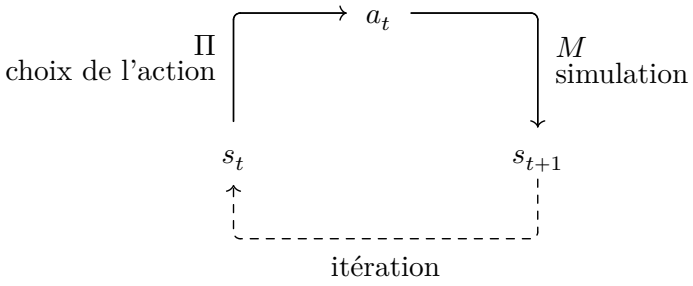
$$\forall (s, a) \in S \times A, \forall r \in \text{récompenses}, r(s, a) := r(s) \tag{3}$$

2.2.1 Chemins d'états possibles $\mathcal{C}$

M et Π forment en fait tout se qui se passe pendant un pas de temps, c'est cette boucle que l'on répète pour soit entraîner l'agent (si l'on met Π à jour à chaque tour de boucle) ou l'utiliser:

[3] i.e. $\text{card}\, \rho_0(S) = 1$

[4] En pratique, $\mathbb{R}$ est discrétisé dans les simulateurs numérique, donc cette hypothèse ne pose pas de problèmes à l'application de la théorie au domaine de la robotique



Quand on « déroule » Π en en partant d'un certain état initial s_0, on obtient une suite d'états et d'actions:

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots$$

Pour tout pas de temps $t \in \mathbb{N}$, on a:

$$\begin{cases} a_t &= \Pi(s_t) \\ s_{t+1} &= M(s_t, a_t) \end{cases} \tag{4}$$

Un chemin se modélise aisément par une suite d'éléments de[5] $S \times A$. Ainsi, on note

$$\mathcal{C}_\pi := \left\{ (s_t, a_t)_{t\in\mathbb{N}} \text{ avec } \left\{ \begin{array}{lll} & a_0 &= \pi(s_0) \\ \forall t \in \mathbb{N} & a_{t+1} &= \pi(s_t) \\ \forall t \in \mathbb{N} & s_{t+1} &= M(s_t, a_t) \end{array} \right. \middle| \; s_0 \in S \right\} \tag{5}$$

l'ensemble des chemins possibles avec la politique π. C'est tout simplement l'ensemble de tout les « déroulements » de la politique π en partant des états possibles de l'environnement.

On définit également l'ensemble de *tout* les chemins d'états possibles, peut importe la politique, $\mathcal{C}$:

$$\mathcal{C} := \left\{ \left\{ \begin{array}{lll} & c_0 &= (s_0, a_0) \\ \forall t \in \mathbb{N} & c_{t+1} &= (M(c_t), a_t) \end{array} \right. \middle| \; (s_0, a) \in S \times A^{\mathbb{N}} \right\} \tag{6}$$

On notera que, selon M, on peut avoir $\mathcal{C} \subsetneq (S \times A)^{\mathbb{N}}$: par exemple, certains états de l'environnement peuvent représenter des « impasses », où il est impossible d'évoluer vers un autre état, peut importe l'action choisie.

[5] Il est essentiel de conserver l'information de l'action prise entre chaque état (contrairement à ce que fournirait une simple suite d'éléments de S, par exemple) pour pouvoir calculer de probabilités par rapport à une politique le long de ce chemin. En effet, on peut savoir avec quelle probabilité Π choisit une certaine action $a \in A$ depuis un certain $s \in S$, mais encore faut-il savoir « par quelle $a \in A$ est-on passé ».

On note aussi que $\mathcal{C}$ (et donc $\mathcal{C}_\pi$ aussi) est dénombrable, étant construit à partir de $(S \times A)^{\mathbb{N}}$ et S, A et $\mathbb{N}$ étant aussi dénombrables[6]

Cette formalisation est utile par la suite,
pour proprement définir certaines grandeurs.

Remarque

Les définitions suivantes, dont la plupart proviennent du papier *Trust Region Policy Optimization*, citation « [7] », ont été reformulées pour utiliser cette notion de chemins.

Notamment, les espérances le long d'un chemin, notées $\mathbb{E}_{s_0,a_0,\dots}$ dans [7], sont dénotées ici par une opération-sur-ensemble usuelle[7], avec $\mathop{\mathbb{E}}_{c\in\mathcal{C}}$. De même, la notation $\mathbb{E}_{s_0,a_0,\dots\sim\pi}$ est dénotée $\mathop{\mathbb{E}}_{c\sim\pi\in\mathcal{C}}$ et explicitée après (8).

Dans la documentation de *OpenAI Spinning Up* (citation « [8] »), les espérances sont notées $\mathop{E}_{s,a\sim\pi}$, ce qui correspond à faire une espérance le long de tout chemin: cela correspond ici à $\mathop{\mathbb{E}}_{c\sim\pi\in\mathcal{C}} \sum_{t=0}^{\infty} \cdots$.

2.2.2 Récompense attendue η

η représente la récompense moyenne à laquelle l'on peut s'attendre pour une politique π avec fonction de récompense r.

Elle prend en compte le *discount factor* γ : les récompenses des actions deviennent de moins en moins importantes avec le temps. η est définie ainsi [7]

$$\eta(\pi, r) := \underbrace{\sum_{(c_t)_{t\in\mathbb{N}}\in\mathcal{C}} \underbrace{\rho_0(s_0) \prod_{t=0}^{\infty} Q_\pi(c_t)}_{\text{probabilité du chemin}} \quad \underbrace{\sum_{t=0}^{\infty} \gamma^t r(c_t)}_{\text{récompense associée}}}_{\text{pour tout chemin possible}} \tag{7}$$

On peut également exprimer $\eta(\pi, r)$ comme une espérance. On a (cf preuve en A.2)

$$\eta(\pi, r) = \mathop{\mathbb{E}}_{C\sim\pi\in\mathcal{C}} \left(\sum_{t=0}^{\infty} \gamma^t r(C_t) \right) \tag{8}$$

Avec $C \sim \pi \in \mathcal{C}$ signifiant

- C est une variable aléatoire à valeur dans $\mathcal{C}$
- C sui la même loi que π

2.2.3 Avantage A

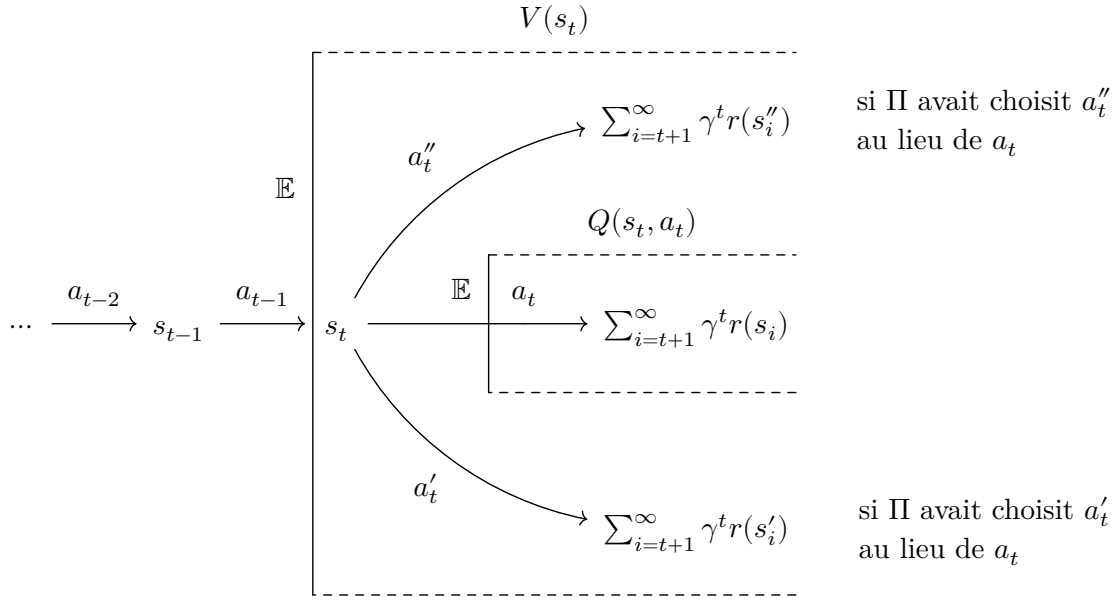
L'avantage $A_{\pi,r}(s, a)$ mesure à quel point il est préférable de choisir l'action a quand on est dans l'état s (pour la politique π, avec « préférable » au sens de $\mathop{\geq}_{r(M(s,\cdot))}$[8])

[6] On a $\operatorname{card}\mathcal{C} \leq \operatorname{card}((S \times A)^{\mathbb{N}}) = \operatorname{card}(S \times A)^{\operatorname{card}\mathbb{N}} = (\operatorname{card} S \operatorname{card} A)^{\operatorname{card}\mathbb{N}} \leq (\aleph_0)^{\operatorname{card}\mathbb{N}} = {}^2\aleph_0 = \aleph_0$

[7] d'autres exemples d'« opérations-sur-ensemble » sont $\sum_{x\in\mathbb{R}}$ ou $\prod_{n\in\mathbb{N}}$, par exemple. L'« espérance-sur-ensemble » est définie par le passage de (7) à (8)

[8] En posant, pour toute fonction $f : I \to O$, avec O ordonné par $\geq$: $\forall i \in I^2, \quad i_1 \geq_f i_2 := f(i_1) \geq f(i_2)$. Ici donc, on compare les politiques selon $a \mapsto r(M(s, a))$. Autrement dit, la récompense associé à l'état obtenu après le choix d'une action, depuis l'état s

On peut visualiser ce calcul ainsi:



Pour calculer $A_{\pi,r}(s,a)$, on regarde l'espérance des récompenses cumulées pour tout chemin commençant par s, et on la compare à celle pour tout chemin commençant par $M(s,a)$

$$A_{\pi,r}(s,a) := \underbrace{\mathop{\mathbb{E}}_{\substack{(s_t,a_t)_{t\in\mathbb{N}} \sim \pi \in \mathcal{C} \\ s_0=s \\ s_1=M(s_0,a)}} \sum_{t=0}^{\infty} \gamma^t r(s_t)}_{Q(s,a)} - \underbrace{\mathop{\mathbb{E}}_{\substack{(s_t,a_t)_{t\in\mathbb{N}} \sim \pi \in \mathcal{C} \\ s_0=s}} \sum_{t=0}^{\infty} \gamma^t r(s_t)}_{V(s)} \tag{9}$$

On considère tout les chemins à partir de l'état s_t, et l'on regarde l'espérance…

pour $V(s_t)$ de tout les chemins

pour $Q(s_t, a_t)$ du chemin où l'on a choisi a_t

En suite, il suffit de faire la différence, pour savoir l'*avantage* que l'on a à choisir a_t par rapport au reste.

2.2.4 Lien entre η et A

Pour une fonction de récompense r donnée, A permet de calculer η pour une politique π en fonction de la valeur de η pour une autre politique π' [7]

$$\eta(\pi', r) = \eta(\pi, r) + \mathop{\mathbb{E}}_{(c_t)_{t\in\mathbb{N}} \sim \pi' \in \mathcal{C}} \sum_{t=0}^{\infty} \gamma^t A_{\pi,r}(c_t) \tag{10}$$

2.2.5 *Surrogate advantage* $\mathcal{L}$

Il est théoriquement possible d'utiliser A pour optimiser une politique, en maximisant sa valeur à un état donné:

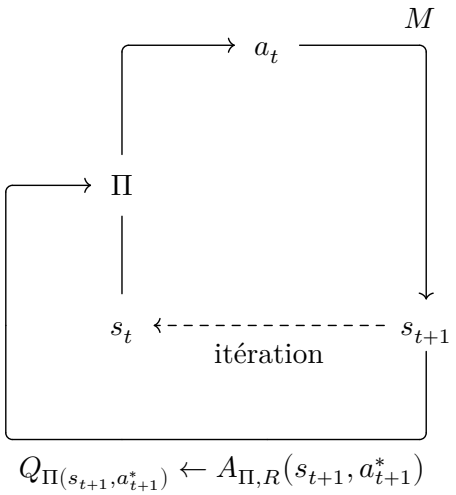


Fig. 5. – Boucle d'entraînement

Avec

$$a^*_{t+1} \coloneqq \underset{a\in A}{\operatorname{argmax}}\, A_{\Pi,R}(s_{t+1},a) \tag{11}$$

Mais, en pratique, des erreurs d'approximations peuvent rendre $A_{\Pi,R}(s_{t+1},a^*_{t+1})$ négatif, ce qui empêche de s'en servir pour définir une valeur de Q_Π [7]

Le *surrogate advantage* détermine la performance d'une politique par rapport à une autre [8]

$$\mathcal{L}_r(\pi',\pi) \coloneqq \underset{(s_t,a_t)_{t\in\mathbb{N}}\in\mathcal{C}}{\mathbb{E}} \sum_{t=0}^{\infty} \frac{Q_\pi(s_t,a_t)}{Q_{\pi'}(s_t,a_t)} A_{\pi,r}(s_t,a_t) \tag{12}$$

2.3 Méthodes d'optimisation de politique

2.3.1 *Trust Region Policy Optimization*

La méthode TRPO définit la mise à jour de Q avec un Q' qui maximise le *surrogate advantage* [8], sous une contrainte limitant l'écart entre Q et Q'

L'idée de la *TRPO* est de maximiser le *surrogate advantage* du nouveau Q tout en limitant l'ampleur des modifications apportées à Q, ce qui procure une stabilité à l'algorithme, et évite qu'un seul « faux pas » dégrade violemment la performance de la politique.

$$Q' = \begin{cases} \arg\max_q \mathcal{L}_r(q, Q) \\ \text{s.c. distance}(Q', Q) < \delta \end{cases} \tag{13}$$

Avec δ une limite supérieure de distance entre Q', la nouvelle politique, et Q, l'ancienne.

Distance entre politiques

Il existe plusieurs manières de mesurer l'écart entre deux distributions de probabilité, dont notamment la *divergence de Kullback-Leibler*, aussi appelée entropie relative [9], [10]:

$$D_{\mathrm{KL}}(P \parallel P') \coloneqq \sum_{x \in \mathcal{X}} P(x) \log \frac{P(x)}{P'(x)} \tag{14}$$

Avec $\mathcal{X}$ l'espace des échantillons et P, P' deux distributions de probabilité sur celui-ci. Dans notre cas, $\mathcal{X} = S \times A$,

Pour évaluer cette distance, on regarde la plus grande des distances entre des paires de distributions de probabilité de politiques Q_Π et $Q_{\Pi'}$ pour $s \in S$ fixé [7]

$$\max_{s \in S} D_{\mathrm{KL}}(Q_{\Pi'}(s, \cdot) \parallel Q_\Pi(s, \cdot)) < \delta \tag{15}$$

En notant $Q_\pi(s, \cdot) \coloneqq a \mapsto Q_\pi(s, a)$. On a donc ici « $\mathcal{X} = A$ » dans la définition de D_{KL}

Pourquoi faire le maximum sur chaque $s \in S$?

Ce maximum revient à limiter non pas la simple distance entre les deux politiques, mais *limiter la modification de la politique sur chacune de ses actions.*

Ceci permet d'éviter d'avoir deux politiques jugées similaires par D_{KL} à cause de modifications se « compensant ». Par exemple, avec

$$\forall s \in S, Q(s, 1) = Q(s, 2) \tag{16}$$

et

$$Q' \coloneqq (s, a) \mapsto \begin{cases} Q(s, a) \cdot 2 & \text{si } a = 1 \\ Q(s, a) \cdot \frac{1}{2} & \text{si } a = 2 \\ Q(s, a) & \text{sinon} \end{cases} \tag{17}$$

On a $D_{\mathrm{KL}}(Q, Q') = 0$ (cf preuve en A.1), alors qu'il y a eu une modification très importante des probabilités de choix de l'action 1 et 2 dans tout les états possibles : si on imagine $Q(s, 1) = Q(s, 2) = 1/4$, on a après modification $Q'(s, 1) = 1/2$ et $Q'(s, 2) = 1/8$.

Région de confiance

Cette contrainte définit un ensemble réduit de Π' acceptables comme nouvelle politique, aussi appelé une *trust region* (région de confiance), d'où la méthode d'optimisation tire son nom [7].

En pratique, l'optimisation sous cette contrainte est trop demandeuse en puissance de calcul, on utilise plutôt l'espérance [7]

$$\overline{D_{\mathrm{KL}}} \coloneqq \mathbb{E}_{s\in S} D_{\mathrm{KL}}(Q(s,\cdot) \parallel Q'(s,\cdot)) \tag{18}$$

2.3.2 *Proximal Policy Optimization*

La *PPO* repose sur le même principe de stabilisation de l'entraînement par limitation de l'ampleur des changements de politique à chaque pas.

Cependant, les méthodes *PPO* préfèrent changer la quantité à optimiser, pour limiter intrinsèquement l'ampleur des modifications, en résolvant un problème d'optimisation sans contraintes [11]

$$\begin{aligned} &\underset{\Pi'}{\arg\max} \underset{(s,a)\in\mathcal{C}}{\mathbb{E}} L(s,a,\Pi,\Pi',R) \\ &\text{s.c. } \top \end{aligned} \tag{19}$$

Avec pénalité *(PPO-Penalty)*

PPO-Penalty soustrait une divergence K-L pondérée à l'avantage:

$$L(s,a,\Pi,\Pi',R) = \frac{Q_\Pi(s,a)}{Q_{\Pi'}(s,a)} A_{\Pi,R}(s,a) - \beta D_{\mathrm{KL}}(\Pi \parallel \Pi') \tag{20}$$

Avec β ajusté automatiquement pour être dans la même échelle que l'autre terme de la soustraction.

Par *clipping (PPO-Clip)*

PPO-Clip utilise une limitation du ratio de probabilités (en minimum et en maximum) [12]

$$\begin{aligned} L(s,a,\Pi,\Pi',R) = \min\Bigg(&\frac{Q_{\Pi'}(s,a)}{Q_\Pi(s,a)} A_{\Pi',R}(s,a), \\ &\mathrm{clip}\left(\frac{Q_{\Pi'}(s,a)}{Q_\Pi(s,a)}, 1-\varepsilon, 1+\varepsilon\right) A_{\Pi',R}(s,a)\Bigg) \end{aligned} \tag{21}$$

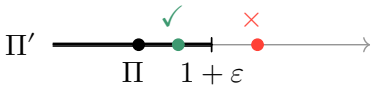
Avec $\varepsilon \in \mathbb{R}_+^*$ un paramètre indiquant à quel point l'on peut s'écarter de la politique précédente, et

$$\mathrm{clip} \coloneqq (x,m,M) \mapsto \begin{cases} m & \text{si } x < m \\ M & \text{si } x > M \\ x & \text{sinon} \end{cases} \tag{22}$$

La complexité de l'expression, et la présence d'un min au lieu de simplement un clip est dûe au fait que l'avantage $A_{\Pi',R}(s,a)$ peut être négatif. L'expression se simplifie en séparant les cas (cf preuve en A.3)

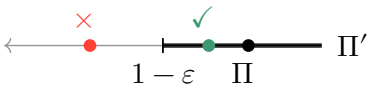
Si l'avantage est positif a est un meilleur choix que $\Pi(s)$.

$$L(s,a,\Pi,\Pi',R) = \min\left(\frac{Q_{\Pi'}(s,a)}{Q_{\Pi}(s,a)},\quad 1+\varepsilon\right) A_{\Pi',R}(s,a)$$



Si l'avantage est négatif choisir a est pire que garder $\Pi(s)$.

$$L(s,a,\Pi,\Pi',R) = \max\left(1-\varepsilon,\quad \frac{Q_{\Pi'}(s,a)}{Q_{\Pi}(s,a)}\right) A_{\Pi',R}(s,a)$$



2.4 CaT (*Constraints as Terminations*)

2.5 Application en robotique

Dans le contexte de la robotique, le calcul de l'état post-action de l'environnement est le travail du *moteur de physique*.

Bien évidemment, ce sont des programmes complexes avec des résolutions souvent numériques d'équation physiques; il est presque inévitable que des bugs se glissent dans ces programmes.

Un environnement de RL[9] ne se résume pas à son moteur de physique: il faut également charger des modèles 3D, le modèle du robot (qui doit être contrôlable par les actions, on fait donc une émulation de la partie logicielle du robot), et également, pendant les phases de développement, avoir un moteur de rendu graphique, une interface et des outils de développement.

Cet ensemble s'appelle un *simulateur système*.

2.5.1 Spécification de la tâche

Le score (récompense ou coût) dépend de la tâche pour laquelle on veut entraîner l'agent.

En robotique, il est commun d'inclure dans la récompense les éléments suivants:

- Couple maximal sur les commandes envoyées au moteurs
- Limite sur la vélocité du robot
- Prévention des auto-collisions (par exemple, le bras qui tape la jambe)
- (TODO: Ajouter [13])
- etc.

2.5.2 Inventaire des simulateurs en robotique

Isaac

Un simulateur développé par NVIDIA [14], utilisant son propre moteur de rendu, PhysX [15]

MuJoCo

Un simulateur initialement propriétaire. Il a été rendu gratuit puis open source par Google DeepMind [16].

Bien que MuJoCo est décrit comme un moteur de simulation physique et non un simulateur, il embarque une commande `simulate` qui le rend fonctionnellement équivalent à un simulateur [17].

Gazebo

[9] Reinforcement Learning

Les intérêts de Gazebo [18] sont multiples:

- C'est un logiciel open-source *communautaire*, qui ne dépend pas du financement d'une grande entreprise
- Son architecture modulaire permet notamment d'utiliser plusieurs moteurs de simulation physique différents [19], à l'inverse de MuJoCo.
- C'est un *simulateur système*, qui est capable de simuler la partie logicielle du robot en plus de la physique du son modèle 3D.

Gazebo possède des plugins officiels pour divers moteurs de simulation physique:

DART Plugin `gz-physics-dartsim-plugin`, c'est l'implémentation principale, et celle par défaut [19], [20].

Bullet Plugin `gz-physics-bulletsim-plugin`. En beta [19], [21], [22].

Bullet Featherstone Plugin `gz-physics-bullet-featherstone-plugin`, également en beta [19]. Une variable de Bullet, utilisant l'algorithme de Featherstone [23], [24]

2.6 Le robot *H1v2* d'Unitree

H1v2 est un modèle de robot humanoïde créé par la société Unitree.

Il possède plus de 26 degrés de liberté, dont

- 6 dans chaque jambe (3 à la hanche, 2 au talon et un au genou),
- 7 dans chaque bras (3 à l'épaule, 3 au poignet et un au coude) [25]

2.7 Reproductibilité logicielle

La reproductibilité est particulièrement complexe dans le champ du reinforcement learning [26].

En plus des difficultés de reproductibilité sur l'algorithme lui-même, le paysage logiciel et matériel est riche en dépendances à des bibliothèques, qui elle aussi dépendent d'autres bibliothèques.

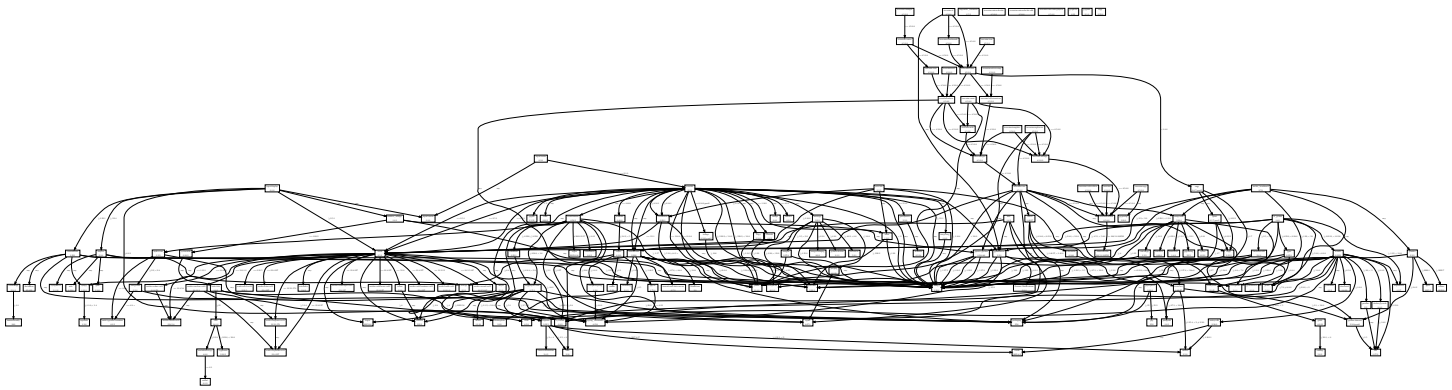


Fig. 8. – Arbre des dépendances pour *Gepetto/h1v2-Isaac*

Bien que toutes ces dépendances puissent être spécifiées à des versions strictes [27] pour éviter des changements imprévus de comportement du code venant des bibliothèques, beaucoup celles-ci ont besoin de compiler du code C++ à l'installation pour des raisons de performance [28].

Des problèmes de reproductibilité peuvent donc subsister à l'installation des dépendances, étant donné la dépendance du processus de compilation à la machine compilant le code.

3 Packaging reproductible avec Nix

3.1 Reproductibilité

3.1.1 État dans le domaine de la programmation

La différence entre une fonction au sens mathématique et une fonction au sens programmatique consiste en le fait que, par des raisons de practicité, on permet aux `functions` des langages de programmation d'avoir des *effets de bords*. Ces effets affectent, modifient ou font dépendre la fonction d'un environnement global qui n'est pas explicitement déclaré comme une entrée (un argument) de la fonction en question [29].

Cette liberté permet, par exemple, d'avoir accès à la date et à l'heure courante, interagir avec un système de fichier d'un ordinateur, générer une surface pseudo aléatoire par bruit de Perlin, etc.

Mais, en contrepartie, on perd une équation qui est fondamentale en mathématiques:

$$\forall E, F, \forall f : E \to F, \forall (e_1, e_2) \in E^2, e_1 = e_2 \Rightarrow f(e_1) = f(e_2) \tag{23}$$

En programmation, on peut très facilement construire un f qui ne vérifie pas ceci:

```python
from datetime import date

def f(a):
  return date.today().year + a
```

Selon l'année dans laquelle nous sommes, `f`(0) n'a pas la même valeur.

De manière donc très concrète, si cette fonction `f` fait partie du protocole expérimental d'une expérience, cette expérience n'est plus reproductible, et ses résultats sont donc potentiellement non vérifiables, si le papier est soumis le 15 décembre 2025 et la *peer review* effectuée le 2 janvier 2026.

3.1.2 Contenir les effets de bords

En dehors du besoin de vérifiabilité du monde de la recherche, la reproductibilité est une qualité recherchée dans certains domaines de programmation [30]

Il existe donc depuis longtemps des langages de programmation dits *fonctionnels*, qui, de manière plus ou moins stricte, limite les effets de bords. Certains langages font également la distinction entre une fonction *pure*[10] et une fonction classique [31]. Certaines fonctions, plutôt appelées *procédures*, sont uniquement composées d'effet de bord puisqu'elle ne renvoie pas de valeur [32]

3.1.3 État dans le domaine de la robotique

En robotique, pour donner des ordres au matériel, on intéragit beaucoup avec le monde extérieur (ordres et lecture d'état de servo-moteurs, flux vidéo d'une caméra, etc), souvent dans un langage plutôt bas-niveau, pour des questions de performance et de proximité abstractionnelle au matériel.

De fait, les langages employés sont communément C, C++ ou Python[11] [33], des langages bien plus impératifs que fonctionnels [34].

[10] sans effets de bord

[11] Il arrive assez communément d'utiliser Python, un langage haut-niveau, mais c'est dans ce cas à but de prototypage. Le code contrôlant les moteurs est écrit dans un langage bas niveau, mais appelé par Python via FFI.

L'idée de s'affranchir d'effets de bords pour rendre les programmes dans la recherche en robotique reproductibles est donc plus utopique que réaliste.

3.1.4 Environnements de développement

Cependant, ce qui fait un programme n'est pas seulement son code: surtout dans des langages plus anciens sans gestion de dépendance intégrée au langage, les dépendances (bibliothèques) du programme, ainsi que l'environnement et les étapes de compilation de ce dernier, représentent également une partie considérable de la complexité du programme (par exemple, en C++, on utilise un outil générant des fichiers de configuration pour un autre outil qui à son tour configure le compilateur de C++ [35])

C'est cette partie que Nix, le gestionnaire de paquet, permet d'encapsuler et de rendre reproductible. Dans ce modèle, la compilation (et de manière plus générale la construction, ou *build*) du projet est la fonction que l'on veut rendre pure. L'entrée est le code source, et le résultat de la fonction est un binaire, qui ne doit dépendre que du code source.

$$\forall\ \text{src}, \text{bin}, \forall f \in \text{bin}^{\text{src}}, \forall (P_1, P_2) \in \text{src}^2, P_1 = P_2 \Rightarrow f(P_1) = f(P_2) \tag{24}$$

Ici, P_1 et P_2 sont deux itérations du code source (src) du programme. Si le code source est identique, les binaires résultants de la compilation (f) sont égaux, au sens de l'égalité bit à bit.

On a la proposition (1), avec $E = \text{src}$, l'ensemble des code source possibles pour un langage, et $F = \text{bin}$, l'ensemble des binaires éxécutables

Nix ne peut pas garantir que le programme sera sans effets de bords au *runtime*, mais vise à le garantir au *build-time*.

3.2 Nix, le gestionnaire de paquets pur

3.2.1 Un *DSL*[12] fonctionnel

Une autre caractéristique que l'on trouve souvent dans la famille de langages fonctionnels est l'omniprésence des *expressions*: quasi toute les constructions syntaxiques forment des expressions valides, et peuvent donc servir de valeur

`def g(x, y):`
`  if y == 5:`
`  x = 6`
`  else:`
`  x = 8`
`  return f(x)`
	`let g x y = f (`
	`  if y = 5 then`
	`  6`
	`  else`
	`  8`
	`)`

Python (`if` et `else` sont des instructions)	**OCaml** (`if` et `else` forment une expression)

Afin de décrire les dépendances d'un programme, l'environnement de compilation, et les étapes pour le compiler (en somme, afin de définir le $f \in \text{bin}^{\text{src}}$), Nix comprend un langage d'expressions [36]. Un fichier `.nix` définit une fonction, que Nix sait exécuter pour compiler le code source.

Expression d'une fonction en Python	En Nix
`lambda f(a): a + 3`	`{ a }: a + 3`

Voici un exemple de définition d'un programme, appelée *dérivation* dans le jargon de Nix:

[12] Domain-Specific Language

```
{
  src-odri-masterboard-sdk,

  lib,
  stdenv,
  jrl-cmakemodules,
  cmake,
  python3Packages,
  catch2_3,
}:

stdenv.mkDerivation {
  pname = "odri_master_board_sdk";
  version = "1.0.7";

  src = src-odri-masterboard-sdk;

  preConfigure = ''
    cd sdk/master_board_sdk
  '';

  doCheck = true;

  cmakeFlags = [
    (lib.cmakeBool "BUILD_PYTHON_INTERFACE" stdenv.hostPlatform.isLinux)
  ];

  nativeBuildInputs = [
    jrl-cmakemodules
    python3Packages.python
    cmake
  ];

  buildInputs = with python3Packages; [ numpy ];

  nativeCheckInputs = [ catch2_3 ];

  propagatedBuildInputs = with python3Packages; [ boost ];
}
```

La dérivation ici prend en entrée le code source (`src-odri-masterboard-sdk`), ainsi que des dépendances, que ce soit des fonctions relatives à Nix même (comme `stdenv.mkDerivation`) pour simplifier la définition de dérivation, ou des dépendances au programmes, que ce soit pour sa compilation ou pour son exécution (dans ce dernier cas de figures, les dépendances sont inclues ou reliées au binaire final)

3.2.2 Un ecosystème de dépendances

Afin de conserver la reproductibilité même lorsque l'on dépend de libraries tierces, ces dépendances doivent également avoir une compilation reproductible: on déclare donc des dépendances à des *packages* Nix, disponibles sur *Nixpkgs* [37].

Parfois donc, écrire un paquet Nix pour son logiciel demande aussi d'écrire les paquets Nix pour les dépendances de notre projet, si celles-ci n'existent pas encore, et cela récursivement. On peut ensuite soumettre nos paquets afin que d'autres puissent en dépendre sans les réécrire, en contribuant à *Nixpkgs* [38]

Pour ne pas avoir à compiler toutes les dépendances soit-même quand on dépend de `.nix` de *nixpkgs*, il existe un serveur de cache, qui propose des binaires des dépendances, Cachix [39]

3.2.3 Une compilation dans un environnement fixé

Certains aspects de l'environnement dans lequel l'on compile un programme peuvent faire varier le résultat final. Pour éviter cela, Nix limite au maximum les variations d'environnement. Par exemple, la date du système est fixée au 0 UNIX (1er janvier 1990): le programme compilé ne peut pas dépendre de la date à laquelle il a été compilé.

Quand le *sandboxing* est activé, Nix isole également le code source de tout accès au réseau, aux autres fichiers du système (ainsi que d'autres mesures) pour améliorer la reproductibilité [40]

Un complément utile: compiler en CI

Pour aller plus loin, on peut lancer la compilation du paquet Nix en *CI*[14], c'est-à-dire sur un serveur distant au lieu de sur sa propre machine. On s'assure donc que l'état de notre machine de développement personnelle n'influe pas sur la compilation, puisque chaque compilation est lancée dans une machine virtuelle vierge [41].

3.3 NixOS, un système d'exploitation à configuration déclarative

Une fois le programme compilé avec ses dépendances, il est prêt à être transféré sur l'ordinateur ou la carte de contrôle embarquée au robot.

Lorsqu'il y a un ordinateur embarqué, comme par exemple une Raspberry Pi [42], il faut choisir un OS sur lequel faire tourner le programme.

Là encore, un OS s'accompagne d'un amas considérable de configuration des différentes parties du système: accès au réseau, drivers,…

Sur les OS Linux classiques tels que Ubuntu ou Debian, cette configuration est parfois stockée dans des fichiers, ou parfois retenue en mémoire, modifiée par l'éxécution de commandes.

C'est un problème assez récurrent dans Linux de manière générale: d'un coup, le son ne marche plus, on passe ½h sur un forum à copier-coller des commandes dans un terminal, et le problème est réglé… jusqu'à ce qu'il survienne à nouveau après un redémarrage ou une réinstallation.

Ici, NixOS assure que toute modification de la configuration d'un système est *déclarée* (d'où l'adjectif « déclaratif ») dans des fichiers de configurations, également écrits dans des fichiers `.nix` [43].

Ici encore, cela apporte un gain en terme de reproductibilité: l'état de configuration de l'OS sur lequel est déployé le programme du robot est, lui aussi, rendu reproductible.

3.4 Packaging Nix pour *gz-unitree*

Le packaging pour Nix de gz-unitree lui-même n'est pas très complexe: il s'agit d'un projet C++ / CMake standard.

Cependant, gz-unitree a deux principales dépendances

- Gazebo lui-même, à travers `gz-sim`, `gz-sensors`, `gz-common`, `gz-plugin`, `gz-cmake`, etc.
- Le SDK2 d'Unitree

En ce qui concerne le SDK2 d'Unitree, une déclaration de paquet Nix a pu être écrite sans trop de soucis, la bibliothèque étant également un projet C++ standard:

[14] Continuous Integration, lit. intégration continue

```
{
  lib,
  stdenv,
  fetchFromGitHub,
  cmake,
  eigen,
}:

stdenv.mkDerivation rec {
  pname = "unitree-sdk2";
  version = "2.0.0";

  src = fetchFromGitHub {
    owner = "unitreerobotics";
    repo = "unitree_sdk2";
    rev = version;
    hash = "sha256-r05zwhZW36+VOrIuTCr2HLf2R23csmnj33JFzUqz62Q=";
  };

  nativeBuildInputs = [ cmake ];

  buildInputs = [ eigen ];

  meta = {
    description = "Unitree robot sdk version 2. https://support.unitree.com/home/zh/
developer";
    homepage = "https://github.com/unitreerobotics/unitree_sdk2";
    license = lib.licenses.bsd3;
    maintainers = with lib.maintainers; [ nim65s ];
    platforms = lib.platforms.unix;
  };
}
```

Par contre, en ce qui concerne Gazebo, la situation est plus complexe: étant un simulateur système, le projet est bien plus conséquent, et donc plus dur à packager.

Il existe plusieurs tentaives de packaging de Gazebo pour Nix:

- Un *overlay* ROS (Robot Operating System), qui inclut notamment Gazebo [44]
- Un package pour nixpkgs (registry officielle de Nix) [45], [46], [47]
- Un outil de génération de paquets Nix à partir de paquets ROS ou Gazebo, développé par Guilhem Saurel au sein de l'équipe Gepetto, *gazebros2nix* [48]

Au début du développement de *gz-unitree*, des essais d'utilisation des paquets Nix pour le développement et la compilation ont été réalisés, mais des erreurs subsistaient, en particulier avec Gazebo.

4 Étude du SDK d'Unitree et du bridge SDK ⇆ MuJoCo

Unitree met à disposition du public un *SDK*[15] permettant de contrôler ses robots (dont le H1v2).

4.1 Canaux DDS

Pour communiquer avec le robot via le réseau, Unitree utilise CycloneDDS, une implémentation par Oracle du standard DDS[16] [49], une technologie de communication bidirectionnelle[17] en temps réel,

[15] Kit de développement logiciel (Software Development Kit)
[16] pour Data Distribution Service
[17] dite « *pub-sub* » pour *publish/subscribe*

standardisée par l'Object Management Group, OMG [50]. Les messages sont envoyées sur le réseau via UDP et IP.

Les données contenues dans chacun des messages sont spécifiées via un autre format, IDL, également standardisé par l'OMG [51].

L'intérêt d'un format indépendant du langage de programmation est que l'on peut générer du code décrivant ces données pour plusieurs langages, ce que fait Unitree en distribuant du code C++ et Python.

Par exemple, les messages permettant de contrôler les moteurs du H1v2 sont définis ainsi

```
struct MotorCmd
{
  uint8 mode;
  float q;
  float dq;
  float tau;
  float kp;
  float kd;
  unsigned long reserve;
};

struct Cmd
{
  uint8 mode_pr;
  uint8 mode_machine;
  MotorCmd motor_cmd[35];
  unsigned long reserve[4];
  unsigned long crc;
};
```

Liste 1. – `LowCmd.idl`, traduit depuis sa conversion en C++ [52]

DDS groupe les mesages dans des *topics*. Les messages sont échangés sur un topic de la manière suivante

Lecture En s'abonnant au topic, on reçoit en temps réel les messages qui sont envoyés dessus

Écriture En publiant des messages sur le topic, on les rend disponibles aux abonnés

CycloneDDS est capable d'un débit d'environ $1\,\mathrm{GB\,s^{-1}}$, pour des messages d'environ 1 kB chacun [53]. On remarque, en pratique, des messages entre 0.9 kB et 1.3 kB dans le cas des échanges commandes/état avec le robot

Et enfin, les *topics* peuvent être isolés d'autres topics via des *domain*s.

4.2 Une base de code partiellement open-source

Le code source du SDK d'unitree est disponible sur Github [54]. Cependant, le dépôt git comprend des fichiers binaires déjà compilés:

```
lib
├── aarch64
│   └── libunitree_sdk2.a
└── x86_64
    └── libunitree_sdk2.a
thirdparty
├── CMakeLists.txt
├── include
│   └── ...
└── lib
    ├── aarch64
    │   ├── libddsc.so
    │   ├── libddsc.so.0 -> libddsc.so
    │   ├── libddscxx.so
    │   └── libddscxx.so.0 -> libddscxx.so
    └── x86_64
        ├── libddsc.so
        ├── libddsc.so.0 -> libddsc.so
        ├── libddscxx.so
        └── libddscxx.so.0 -> libddscxx.so
```

Liste 2. – Résultat de `tree lib thirdparty` dans le dépot git

Compiler le SDK nécéssite l'existance de ces fichiers binaires:

```
63 # Create imported target unitree_sdk2
64 add_library(unitree_sdk2 STATIC IMPORTED GLOBAL)
65 set_target_properties(unitree_sdk2 PROPERTIES
66   IMPORTED_LOCATION ${_IMPORT_PREFIX}/lib/libunitree_sdk2.a
67   INTERFACE_INCLUDE_DIRECTORIES "${_IMPORT_PREFIX}/include;${_IMPORT_PREFIX}/
   include"
68   INTERFACE_LINK_LIBRARIES "ddsc;ddscxx;Threads::Threads"
69   LINKER_LANGUAGE CXX
```

Liste 3. – Extrait de `cmake/unitree_sdk2Targets.cmake`

Ici est défini, via `set_target_properties(... IMPORTED_LOCATION)`, le chemin d'une bibliothèque à lier avec la bibliothèque finale [55].

On confirme ceci en lançant `mkdir build && cd build && cmake ..` après avoir supprimé le répertoire `lib/` :

```
-- The C compiler identification is GNU 13.3.0
-- The CXX compiler identification is GNU 13.3.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Setting build type to 'Release' as none was specified.
-- Current system architecture: x86_64
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD
```

```
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Success
-- Found Threads: TRUE
-- Importing: /home/glebihan/playground/unitree_sdk2/thirdparty/lib/x86_64/
libddsc.so
-- Importing: /home/glebihan/playground/unitree_sdk2/thirdparty/lib/x86_64/
libddscxx.so

CMake Error at CMakeLists.txt:42 (message):
Unitree SDK library for the architecture is not found


-- Configuring incomplete, errors occurred!
```

Les logs montrent aussi que les recettes de compilation dépendent de versions précompilées de LibDDSC et LibDDSCXX, dont le code source semble cependant être fourni avec *unitree_sdk2*:

```
thirdparty/include/dds
├── config.h
├── ddsc
|   ├── dds_basic_types.h
|   ├── dds_data_allocator.h
|   ├── dds_internal_api.h
|   ├── dds_loan_api.h
|   ├── dds_opcodes.h
|   ├── dds_public_alloc.h
...
```

Ces particularités laissent planner quelques doutes sur la nature open-source du code: ces binaires requis sont-ils seulement présent pour améliorer l'expérience développeur en accélererant la compilation, ou « cachent »-ils du code non public?

Ces constats ont motivé une première tentative de décompilation de ces `libunitree_sdk2.a` pour comprendre le fonctionnement du SDK2, via *Ghidra* [56].

Cependant, la découverte de l'existance d'un bridge officiel SDK ⇆ Mujoco [57] a rendu cette piste non nécéssaire.

4.3 Un autre bridge existant: `unitree_mujoco`

Unitree propose un bridge officiel pour utiliser son SDK avec Mujoco.

Le fonctionnement d'un bridge est au final assez similaire, quelque soit le simulateur pour lequel on l'écrit: il s'agit d'envoyer l'état du robot au simulateur, et de réagir quand le simulateur envoie des ordres de commandes.

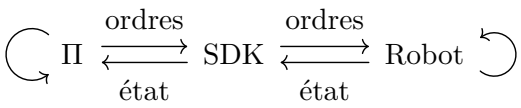


Fig. 9. – Fonctionnement usuel du SDK

Un bridge se substitue au Robot physique, interceptant les ordres du SDK et les traduisants en des appels de fonctions utilisant l'API du simulateur, et symmétriquement pour les envois d'états au SDK. On peut apparenter le fonctionnement d'un bridge à celui d'une attaque informatique de type « Man in the Middle » (MitM).

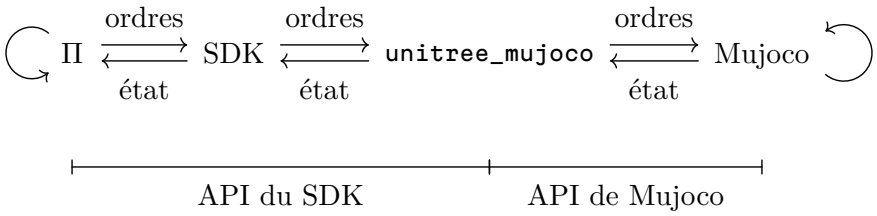


Fig. 10. – Fonctionnement via *unitree_mujoco* du SDK

Le but est de faire la même chose avec notre propre bridge. Le code du bridge Mujoco existant est utile car un bridge, se situant par définition à la frontière entre deux APIs, fait usage des deux APIs.

Écrire un bridge Gazebo pour le même SDK implique donc de changer « API de Mujoco » par « API de Gazebo », mais le code faisant usage du SDK d'Unitree reste le même.

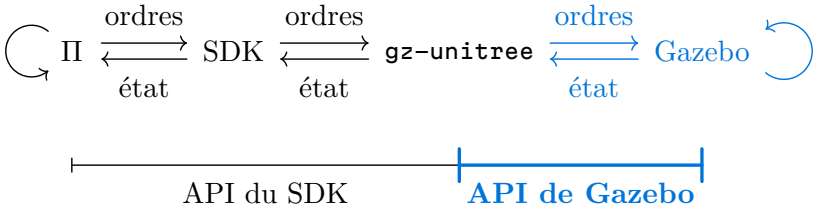


Fig. 11. – Fonctionnement via *gz-unitree* du SDK

Le bridge de Mujoco fonctionne en interceptant les messages sur le canal `rt/lowcmd` et en en envoyant dans le canal `rt/lowstate`, qui correspondent respectivement aux commandes envoyées au robot et à l'état (angles des joints, moteurs, valeurs des capteurs, etc) renvoyé par le robot.

Le `low` indique que ce sont des messages bas-niveau. Par exemple, `rt/lowcmd` correspond directement à des ordres de tension pour les moteurs, au lieu d'envoyer des ordres plus évolués, tels que « se déplacer de x mètres en avant » [58]

Les ordres dans `rt/lowcmd` sont ensuite traduits en appels de fonctions de Mujoco pour mettre à jour l'état du robot simulé, et de messages `rt/lowstate` sont créés à partir des données fournies par Mujoco

Étant donné le modèle *pub/sub* de DDS, on parle de *pub(lication)* de message, et de *sub(scription)*[18] aux messages d'un canal (pour les recevoir)

[18] abonnement

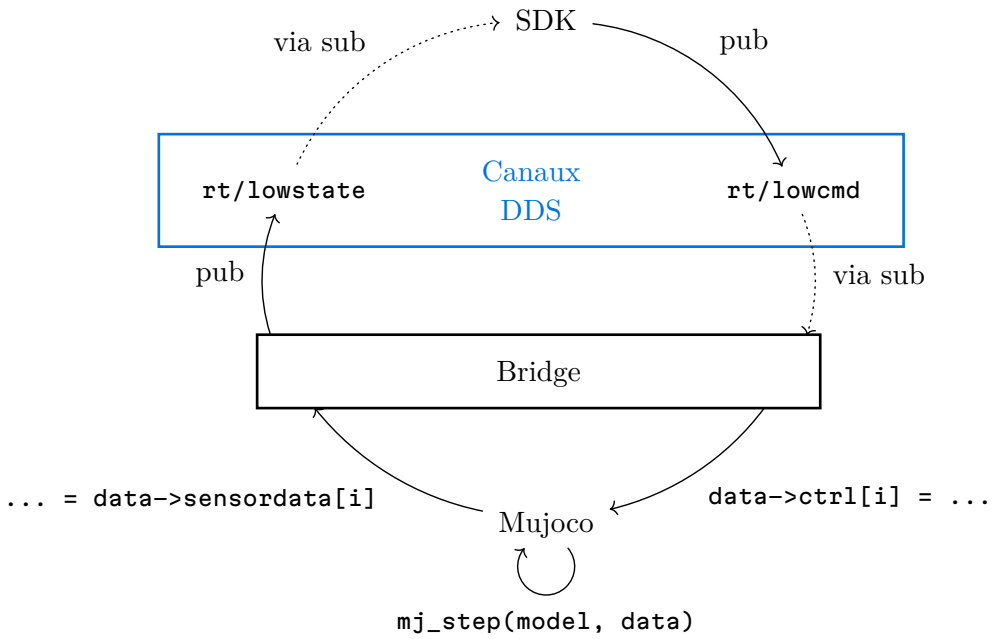


Fig. 12. – Cycle de vie de la simulation avec le bridge pour Mujoco

Le but est donc de reproduire un cycle de vie équivalent, mais en remplaçant la partie spécifique à Mujoco par une partie adaptée à Gazebo.

5 Développement du bridge SDK ⇆ Gazebo

En se basant sur *unitree_mujoco*, il a donc été possible de réaliser un bridge pour Gazebo.

5.1 Établissement du contact

Une première tentative a été de suivre la documentation de CycloneDDS pour écouter sur le canal [59] `rt/lowcmd`, en récupérant les définitions IDL des messages, disponibles sur le dépot `unitree_ros2`[19] [60]

Malheureusement, cette solution s'est avérée infructueuse, à cause d'une inadéquation sur les domaines DDS (ce qui sera compris plus tard).

On change d'approche en préférant plutôt utiliser les abstractions fournies par le SDK de Unitree (cf Chapitre 5.5 et Chapitre 5.6.2)

Enfin, si un pare-feu est actif, il faut autoriser le traffic udp l'intervalle d'addresses IP `224.0.0.0/4`. Par exemple, avec *ufw*

```
sudo ufw allow in proto udp from 224.0.0.0/4
sudo ufw allow in proto udp to 224.0.0.0/4
```

[19] `unitree_mujoco` n'avait pas encore été découvert

Pour arriver à ces solutions, du débuggage du traffic RTPS (le protocole sur lequel est construit DDS [50]), *Wireshark* [61] s'est avéré utile.

C'est notamment grâce à ce traçage des paquets que le problème d'ID de domaine a été découvert: notre *subscriber* DDS était réglé sur le domaine anonyme (ID 0) alors que le SDK d'Unitree communique sur le domaine d'ID 1.

C'est aussi Wireshark qui nous a permis de voir quels étaient les types IDL utilisés pour les messages.

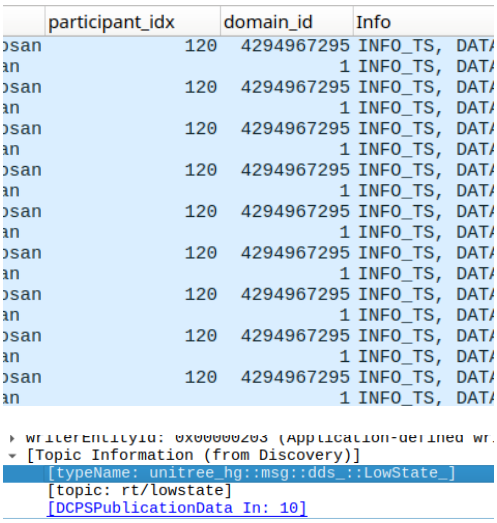


Fig. 13. – *Wireshark* permet de visualiser des méta-données sur les paquets RTPS

Voici une trace wireshark d'un échange usuel entre commandes (`rt/lowcmd`) et états (`rt/lowstate`)

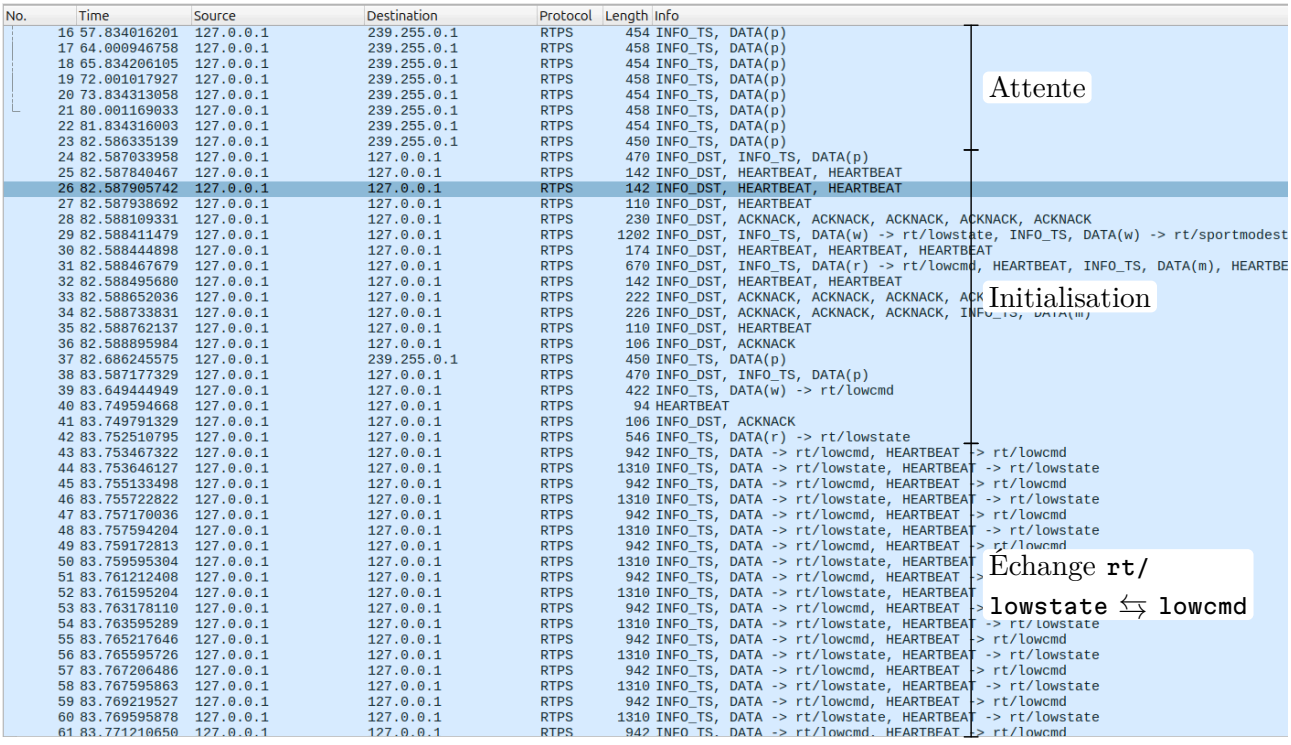


Fig. 14. – Trace de paquets RTPS sur *Wireshark*

5.2 Installation du plugin dans Gazebo

Un *system plugin* Gazebo consiste en la définition d'une classe héritant de `gz::sim::System`, ainsi que d'autres interfaces permettant notamment d'exécuter notre code avant ou après une mise à jour de l'état du simulateur (avec `gz::sim::ISystem{Pre,Post}Update`)

```
#include <gz/sim/System.hh>
namespace gz_unitree
{
    class UnitreePlugin :
        public gz::sim::System,
        public gz::sim::ISystemPreUpdate
    {
    public:
        UnitreePlugin();
    public:
        ~UnitreePlugin() override;
    public:
        void PreUpdate(const gz::sim::UpdateInfo &_info,
                       gz::sim::EntityComponentManager &ecm) override;
    };
}
```

Il faut ensuite implémenter la classe puis appeler une macro ajoutant le plugin à Gazebo

```
#include <gz/plugin/Register.hh>

... // implementation

GZ_ADD_PLUGIN(
    UnitreePlugin,
    gz::sim::System,
    UnitreePlugin::ISystemPreUpdate)
```

Enfin, on active le plugin en le référançant dans le fichier SDF [62], qui décrit l'environnement du simulateurs (objets, éclairage, etc)

```
<sdf version='1.11'>
<world name="default">
  <plugin filename="gz-unitree" name="gz_unitree::UnitreePlugin">
  </plugin>
</world>
<model name='h1_description'>
  <link name='pelvis'>
    <inertial>
    ...
```

Avec `filename` le chemin vers le plugin compilé, qui sera cherché dans les répertoires spécifiés par `GZ_SIM_SYSTEM_PLUGIN_PATH` [63], [64].

5.3 Architecture du plugin

Le plugin consiste en trois parties distinctes:

- Le « branchement » dans les phases de simulation de Gazebo, par l'implémentation de méthodes de `gz::sim::System`
- L'interaction avec les canaux DDS du SDK d'Unitree
- Les données et méthodes internes au plugin

En plus de cela, il y a bien évidemment la politique de contrôle Π, qui interagit via les canaux DDS avec le robot (qu'il soit réel, ou simulé)

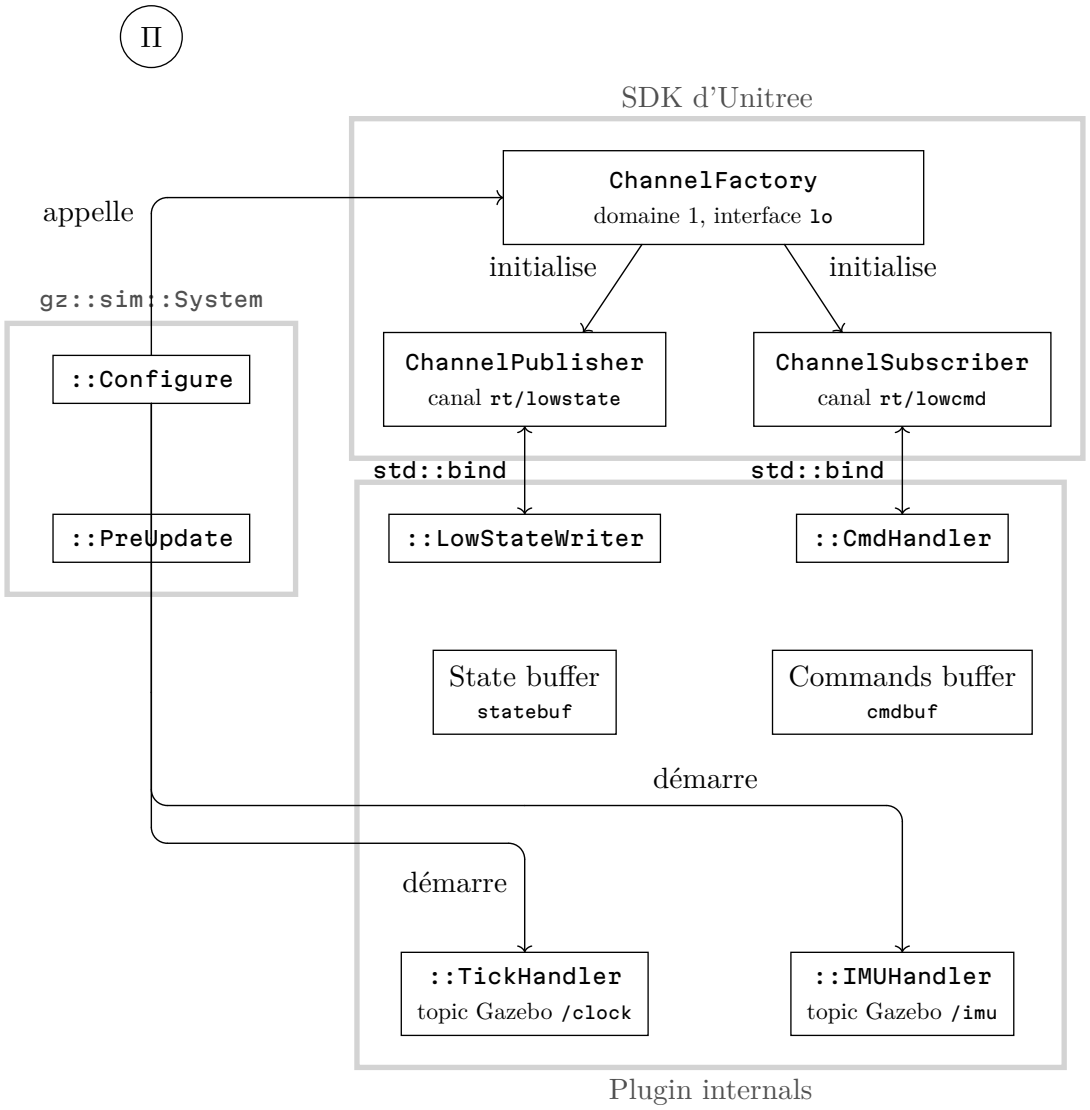


Fig. 15. – Phase d'initialisation du plugin

On commence par instancier un contrôleur dans le domaine DDS n°1, sur l'interface réseau `lo`[20]

On lui associe:

- Un *publisher*, chargé d'envoyer périodiquement des messages sur `rt/lowstate` en appellant la méthode `LowStateWriter`
- Un *subscriber*, chargé d'appeller la méthode `CmdHandler` avec chaque message arrivant sur `rt/lowcmd`.

On démarre aussi deux autres *subscriber*s, qui sont eux chargés d'écouter des messages sur les topics Gazebo `/clock` et `/imu`, ce qui permet de récupérer le tick de simulation et les valeurs du capteur IMU[21], que l'on a préalablement fixé au modèle du robot en le déclarant au fichier SDF chargé par

[20]interface dite « loopback », qui est locale à l'ordinateur: ici, le simulateur et la politique de contrôle tournent sur la même machine, donc les messages DDS n'ont pas besoin de « sortir » de celle-ci

[21]Inertial Measurement Unit, appelée « Centrale intertielle » en français

Gazebo. Le capteur IMU donne des informations importantes sur la position et la vitesse dans l'espace du robot.

Les topics Gazebo sont un autre moyen de communcation inter-processus asynchrone par pub/sub, similaire à DDS [65]. Gazebo utilise Protobuf [66] pour définir les types des messages [67], qui joue ici le même role qu'IDL dans DDS. Les topics Gazebo sont basés sur un réseau de noeuds décentralisé, chaque noeud pouvant publier et/ou recevoir des messages.

Cette initialisation est faite à l'initialisation du plugin par Gazebo, en la faisant dans la méhode `::Configure` du plugin.

En pratique, on utilise `std::bind` [68] pour fixer l'instance d'`UnitreePlugin` et ainsi passer des méthodes de la classe comme des simples fonctions

```
auto subscriber = ChannelSubscriberPtr<LowCmd_>(
  new ChannelSubscriber<LowCmd_>("rt/lowcmd")
);

auto handler = std::bind(
  &UnitreePlugin::CmdHandler,
  this,
  std::placeholders::_1
)

subscriber->InitChannel(handler, 1);

.
```

Liste 4. – Création d'un *subscriber* à `rt/lowcmd` dans `UnitreePlugin::Configure`

```
auto publisher = ChannelPublisherPtr<LowState_>(
  new ChannelPublisher<LowState_>("rt/lowstate")
);

publisher->InitChannel();

this->publisher_thread = CreateRecurrentThreadEx(
  "low_state_writer",
  UT_CPU_ID_NONE,
  500,
  &UnitreePlugin::LowStateWriter,
  this
);
```

Liste 5. – Création du *publisher* pour `rt/lowstate` dans `UnitreePlugin::Configure`

5.4 Calcul des nouveaux couples des moteurs

Pour appliquer une commande à un moteur, on calcule la force effective que le moteur doit appliquer:

$$\tau = \underbrace{\tau_{\text{ff}}}_{\text{stabilité}} + \underbrace{K_p \Delta q}_{\text{propertionnelle}} + \underbrace{K_d \Delta \dot{q}}_{\text{dérivative}} \tag{25}$$

Avec

- $\boldsymbol{\tau}$ pour *torque*, le couple à donner au moteur
- $\boldsymbol{\tau_{\text{ff}}}$ le τ « feed-forward ». Particulièrement utile pour les robots humanoïdes qui doivent rester debout. Dans ces cas, on parle parfois de *gravity compensation part* [69]
- $\boldsymbol{\Delta q}$ écart d'angle de rotation du moteur entre la consigne et l'état actuel
- $\boldsymbol{\Delta \dot{q}}$ vitesse de changement de la consigne[22]
- $\boldsymbol{K_p}$ prépondérance de la partie proportionelle
- $\boldsymbol{K_p}$ prépondérance de la partie dérivée

Cette équation met à jour τ pour rapprocher l'état actuel du moteur de la nouvelle consigne, en prenant en compte

- L'erreur sur l'angle Δq (partie « proportional »)
- L'erreur sur la vitesse de changement de Δq (partie « derivative »)

[22] On a bien $\frac{\mathrm{d}\,\Delta q}{\mathrm{d}\,t} = \Delta \dot{q}$ par linéarité de la dérivation temporelle:

$$\frac{\mathrm{d}\,\Delta q}{\mathrm{d}\,t} = \frac{\mathrm{d}\,q_{\text{new}} - q_{\text{old}}}{\mathrm{d}\,t} = \frac{\mathrm{d}\,q_{\text{new}}}{\mathrm{d}\,t} - \frac{\mathrm{d}\,q_{\text{old}}}{\mathrm{d}\,t} = \Delta \frac{\mathrm{d}\,q}{\mathrm{d}\,t} = \Delta \dot{q} \tag{26}$$

- Un couple dit de *feed-forward*, τ_{ff}, qui permet le maintient du robot à un état stable. On pourrait le déterminer en lançant une première simulation, avec pour objectif le maintient debout. Une fois la stabilité atteinte, on relève les couples des moteurs. Intuitivement, on peut voir τ_{ff} comme un manière de s'affranchir de la partie « maintient debout » dans l'expression de la commande, similairement à la mise à zéro (« tarer ») d'une balance.

Cette prise en compte de la vitesse permet de lisser les changements appliqués aux moteurs

On contrôle la proportion de chaque terme dans le calcul de la nouvelle consigne grâce à deux coefficients, K_p et K_d.

5.5 `rt/lowcmd`

En pratique, les valeurs actuelles pour le calcul de Δq et $\Delta \dot{q}$ proviennent de l'état du moteur, accessible dans `rt/lowstate` avec les champs `q` et `dq` du moteur en question [70]

```
// Avec i l'indice du moteur
auto force = cmdbuf->tau_ff.at(i) +                                  // tau_ff
   cmdbuf->kp.at(i) * (                                              // K_p
     cmdbuf->q_target.at(i) - lowstate.motor_state().at(i).q()       // Delta q
   ) +
   cmdbuf->kd.at(i) * (                                              // K_d
     cmdbuf->dq_target.at(i) - lowstate.motor_state().at(i).dq()     // Delta q.
   );

std::vector<double> torque = {force};
joint.SetForce(ecm, torque);
```

Liste 6. – Implémentation de la mise à jour de τ

Cette équation se rapproche des modèles de type PID (*proportional-integrative-derivative*) [71], avec le terme intégratif remplacé par τ_{ff}, ce qui en fait une expression plus adaptée pour les politiques avec des mouvements non-brusques: le terme intégratif apporte une capacité d'instabilité qui complexifie l'entraînement [Réf. nécéssaire]

Lorsqu'un message, publié par Π (1A) et contenant des ordres pour les moteurs, arrive sur `rt/lowcmd`, `ChannelSubscriber` appelle `::CmdHandler` (2, 3), et modifie un *buffer* (4) contenant la dernière commande reçue.

On trouve dans ces messages les champs nécéssaires à au calcul de τ [72] comme décrit précédemment:

Champ	Type	Description
`crc`	$\mathbb{N}$	Somme de contrôle CRC32, pourrait éventuellement servir à éviter de prendre en compte des messages corrompus
`motor_cmd`	struct.[35]	Paramètres de commande pour chacun des 35 moteurs
`.q`, `.dq`, `.tau`, `.kp` et `.kd`	$\mathbb{R}$	Respectivement q, $\dot{q}$, τ_{ff}, K_p et K_d

Ensuite, Gazebo démarre un nouveau pas de simulation. Avant de faire ce pas, il appelle la méthode `::PreUpdate` sur notre plugin, qui vient chercher la commande stockée dans le *buffer* (1B), et applique cette commande sur le modèle du robot, animé par le simulateur.

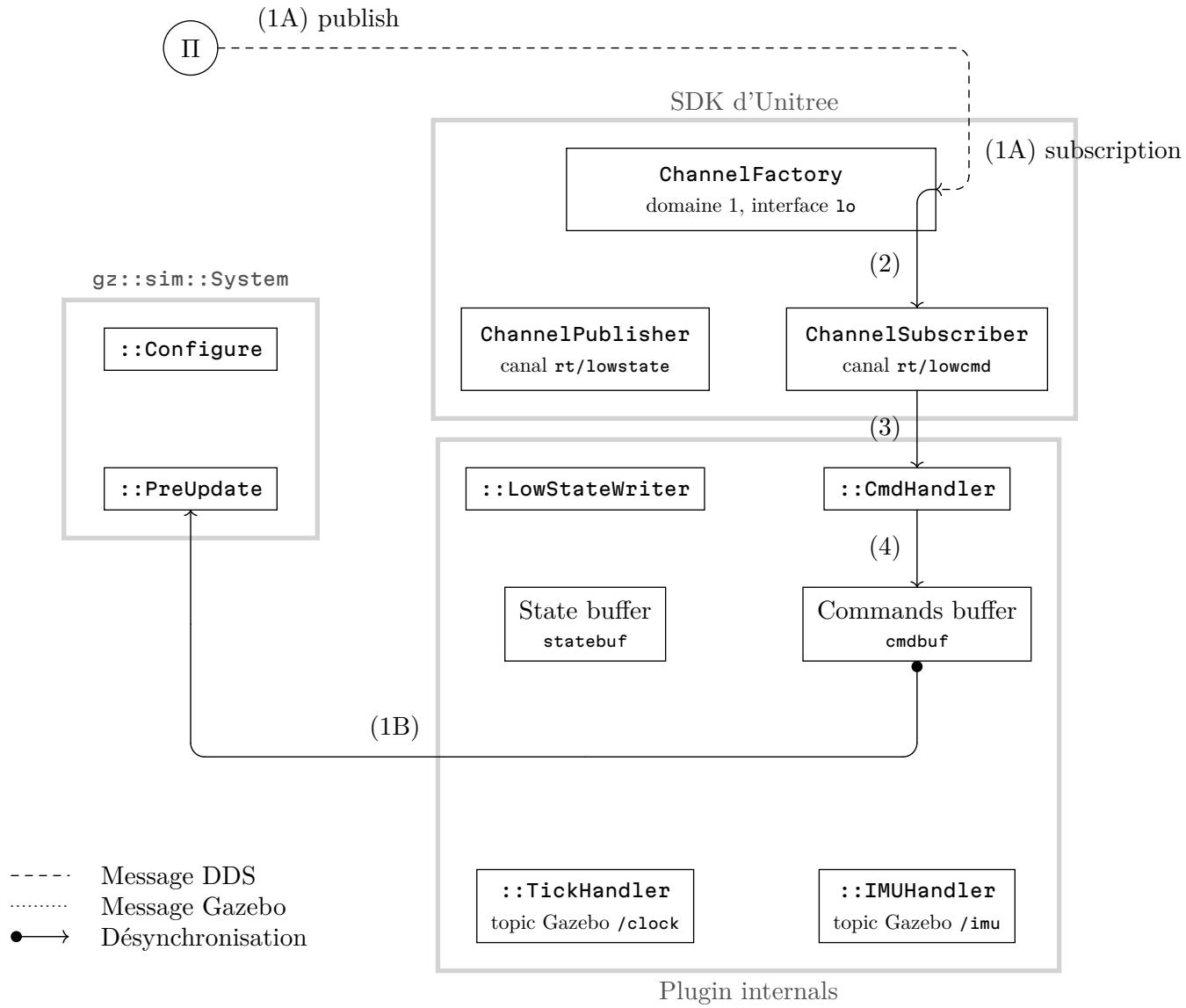


Fig. 16. – Phase de réception des commandes

On notera que (1B) s'exécute *parallèlement* au reste des étapes: la boucle de simulation de Gazebo est indépendante de la boucle de mise à jour de la politique.

Si `::PreUpdate` est plus fréquente Le simulateur appliquera simplement plusieurs fois la même commande, le buffer n'ayant pas été modifié.

Si `::PreUpdate` est moins fréquente Certaines commandes seront simplement ignorées par Gazebo, qui ne vera pas la valeur du buffer avant qu'il change de nouveau.

5.6 `rt/lowstate`

5.6.1 Construction d'un message `rt/lowstate`

La documentation d'Unitree liste l'ensemble des champs disponibles dans un message **`rt/lowstate`**, c'est-à-dire l'ensemble des données que l'on doit récupérer afin de construire nos messages d'état [70]:

Champ	Type	Description	Où récupérer la valeur
`version`	$\mathbb{N}^2$	*Non documenté*	*Laissé vide*
`mode_pr`	$\{0, 1\}$	Défini sur 0 par défaut	0
`mode_machine`	$\{4, 6\}$	Défini sur 6 par défaut	6
`tick`	$\mathbb{N}$ (ms)	*Non documenté*, probablement le temps écoulé depuis le début de la simulation	Messages `gz::msgs::Clock` sur le topic Gazebo `/clock`
`wireless_remote`	$\{0, 1\}^{40}$	*Non documenté*	*Laissé vide*
`reserve`	$\mathbb{N}^4$	*Non documenté*	*Laissé vide*
`crc`	$\mathbb{N}$	Somme de contrôle du message, utilisant *CRC32*.	Implémentation de CRC32 par Unitree[23]
`imu_state…`	struct.	Valeurs des capteurs intertiels du robot	Messages `gz::msgs::IMU` sur le topic Gazebo `/imu` sur le modèle
`.quaternion`	$\mathbb{R}^4$	Posture dans l'espace du robot, dans l'ordre (w, x, y, z)	w, x, y et z sur `.orientation()`
`.rpy`	$\mathbb{R}^3$	Angle d'Euler du robot, dans l'ordre (r, p, y)	`.linear_acceleration()`
`.gyroscope`	$\mathbb{R}^3$	Gyroscope	En utilisant les valeurs de `.orientation()`: $\operatorname{atan}_2(2(wx + yz), 1 - 2(x^2 + y^2)))$ $\operatorname{asin}(2(wy - zx))$ $\operatorname{atan}_2(2(wz + xy), 1 - 2(y^2 + z^2))$
`.accelerometer`	$\mathbb{R}^3$	Accéléromètre	`.angular_velocity()`
`motor_state…`	struct.[35]	Etat de chaque moteur	`gz::sim::Model(…)→joints`
`.mode`	$\{0, 1\}$	0 pour « Brake » et 1 pour « FOC[24] », deux modes de contrôle pour le moteur électrique	0
`.q`	$\mathbb{R}$ (rad)	Angle de rotation du moteur	`.Position()`
`.dq`	$\mathbb{R}$ $(\text{rad} \cdot \text{s}^{-1})$	Angle de rotation du moteur	`.Velocity()`
`.ddq`	$\mathbb{R}$ $(\text{rad} \cdot \text{s}^{-2})$	Angle de rotation du moteur	*Laissé vide*
`.tau_est`	$\mathbb{R}$ $(\text{N} \cdot \text{m})$	Estimation du couple exercé par le moteur	*Laissé vide*

[23] Une implémentation ad-hoc existe dans le code source de `unitree_sdk2` [73] et de `unitree_mujoco` [74], elle est également donnée en annexe B

[24] Field-Oriented Control

5.6.2 Émission de l'état

Avant de démarrer un nouveau pas de simulation, la méthode `::PreUpdate` vient mettre à jour l'état du robot simulé en modifiant le *State buffer* interne au plugin (1A). Gazebo envoie également le nouveau tick de simulation (1C) et les valeurs du capteur IMU (1D) dans leurs topics respectifs.

Le `LowStateWriter` vient lire le *State buffer* (1B) pour publier l'état sur le canal DDS (2, 3) qui est ensuite lu par Π (4), qui (on le suppose) possède une subscription sur `rt/lowstate`.

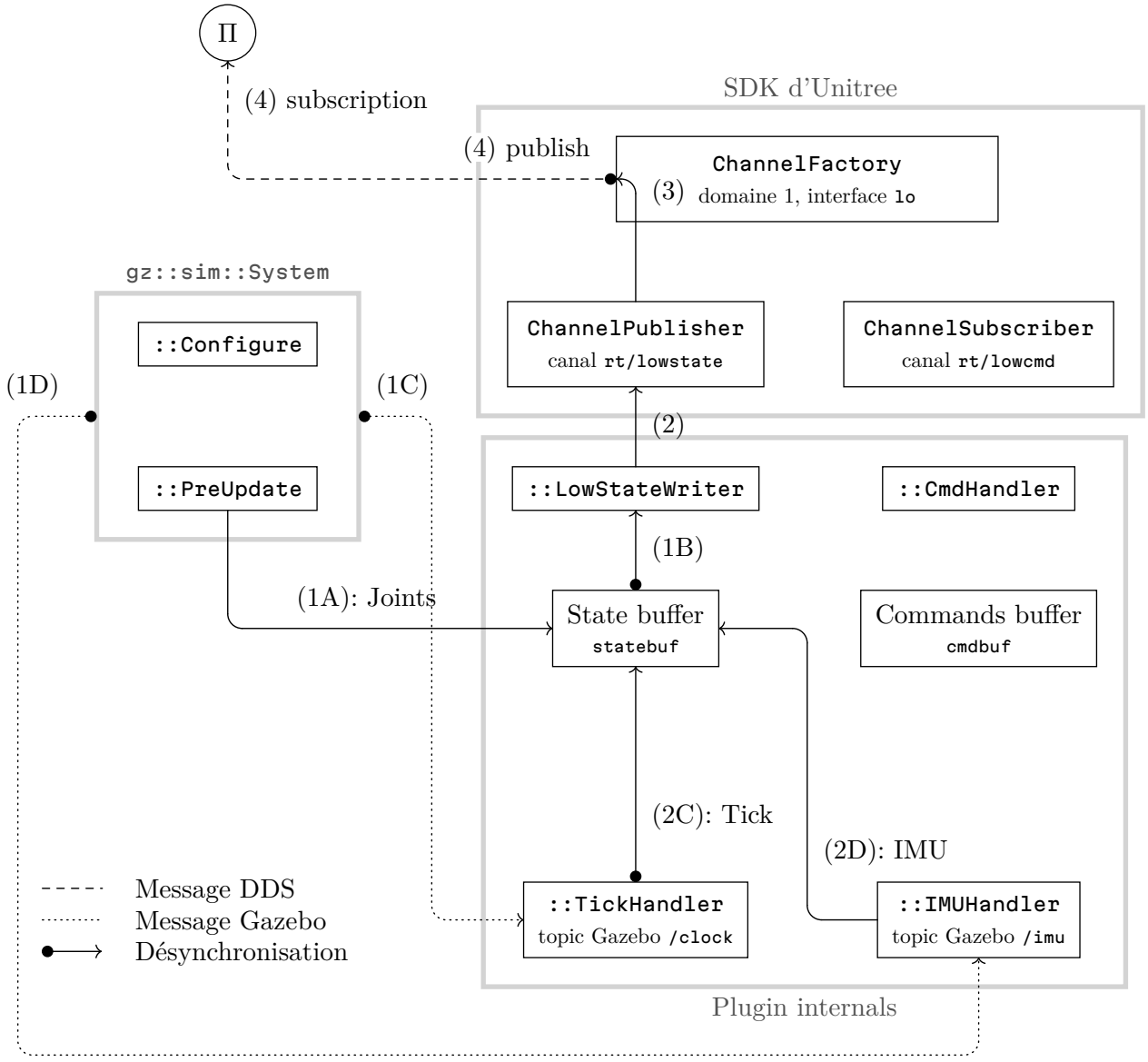


Fig. 17. – Phase d'envoi de l'état

Ici également, `LowStateWriter` s'exécute *en parallèle* du code de `::PreUpdate`: En effet, la création du `ChannelPublisher` démarre une boucle qui vient éxécuter `LowStateWriter` périodiquement, dans un autre *thread*: on a donc aucune garantie de synchronisation entre les deux.

Ici, il y a en plus non pas deux, mais *cinq* boucles indépendantes qui sont en jeux:

- La boucle de simulation de Gazebo (fréquence d'appel de `::PreUpdate`),
- La boucle du `ChannelPublisher` (fréquence d'appel de `::LowStateWriter`), et
- La boucle de réception de Π (fréquence de réception de messages pour Π)
- La boucle de mise à jour du tick (fréquence d'envoi de ticks de simulation par Gazebo)
- La boucle de mise à jour de l'IMU (fréquence d'envoi des valeurs du capteur IMU par Gazebo)

Similairement à la réception de commandes, en comparant à la boucle de mise à jour de Π:

Si `::PreUpdate` est plus fréquente On perdra des états intermédiaires, la résolution temporelle de l'évolution de l'état du robot disponible pour (ou acceptable par[25]) Π sera moins grande

Si `::PreUpdate` est moins fréquente Π reçevra plusieurs fois le même état, ce qui sera représentatif du fait que la simulation n'a pas encore avancé.

On a des effets similaires en comparant la fréquence de la boucle de mise à jour de l'IMU avec celle de la boucle de Π:

Si la boucle IMU est plus fréquente Certaines valeurs du capteur ne seront pas prises en compte par la politique

Si la boucle IMU est moins fréquente Π recevra plusieurs fois le même état, ce qui sera représentatif du fait que la simulation n'a pas encore avancé.

Pour la boucle du tick, cela a peut d'importance. En effet, Π ne dépend probablement pas du tick de simulation, ou si il en dépend, ce serait de manièr peu précise (ce serait plutôt pour savoir « depuis quand est-ce qu'on a lancé la politique », ce qui ne demande pas une précision à la milliseconde) [Réf. nécéssaire]. On met quand même à jour le tick pour que nos messages `rt/lowstate` synthétiques se rapprochent le plus possible des vrais messages tels qu'envoyés par le robot physique.

5.7 Désynchronisations

Dans un même appel de `::PreUpdate`, on effectue d'abord la mise à jour du *State buffer*, puis on lit dans le *Commands buffer*.

Un cycle correspond donc à cinq boucles indépendantes, représentées ci-après:

Bleu Simulation, qui doit englober l'entièreté d'un cycle

Rouge `ChannelPublisher`

Rose Politique Π

Vert Mise à jour de l'IMU

Orange Mise à jour du tick de simulation

[25] En fonction de si `::LowStateWriter` est plus fréquente que Π (dans ce cas là, c'est ce qui est acceptable par Π qui est limitant) ou inversement (dans ce cas, c'est ce que la boucle du publisher met à disposition de Π qui est limitant)

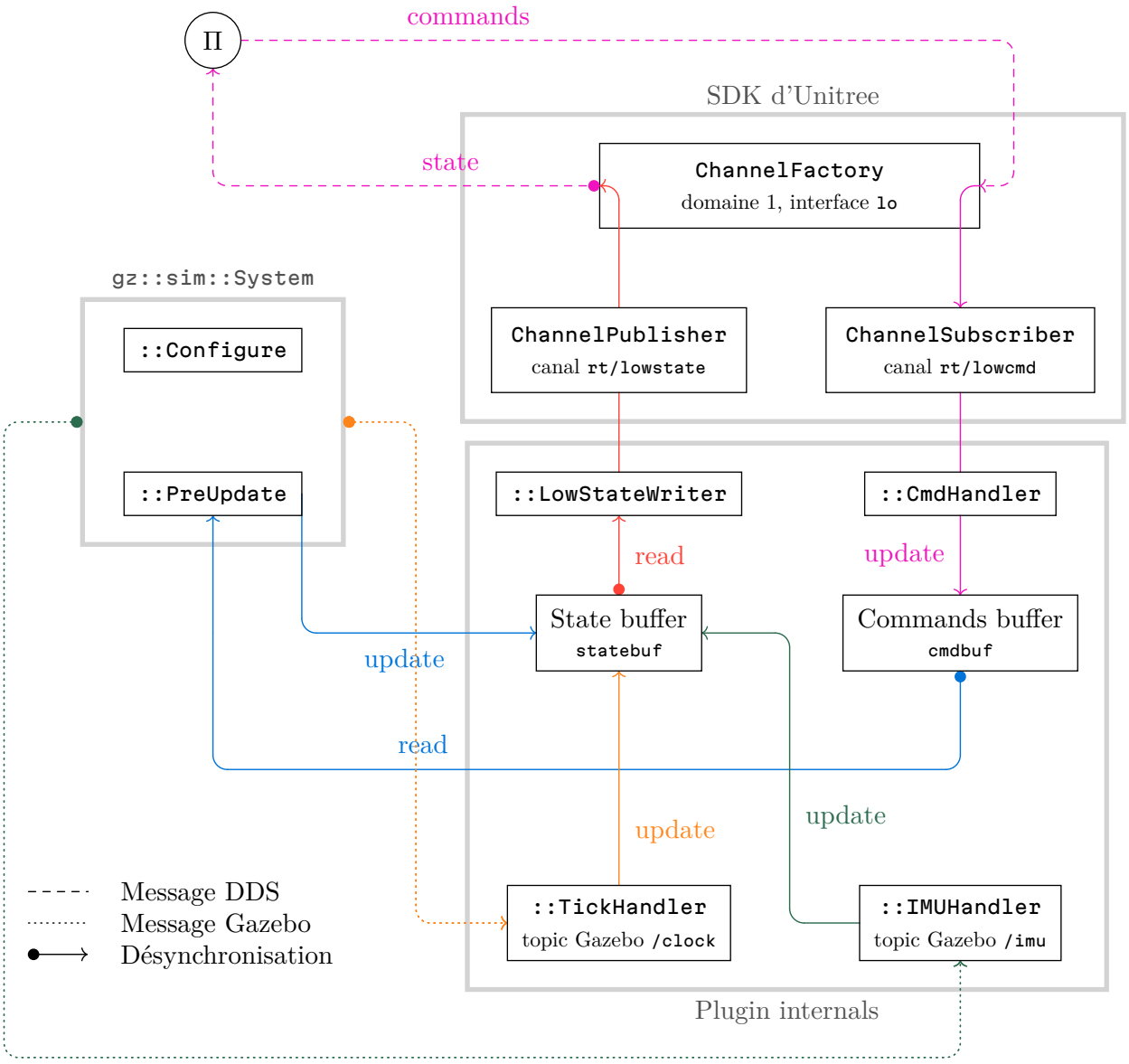


Fig. 18. – Cycle complet. Un cycle commence avec la flèche « update » partant de `::PreUpdate`

Ces désynchronisations pourraient expliquer les problèmes de performance recontrés (cf Chapitre 5.9)

5.8 Vérification sur des politiques réelles

Après avoir testé le bridge sur les politiques d'examples fournies par Unitree, il a été testé sur une politique en cours de développement au sein de l'équipe de robotique du LAAS, Gepetto.

L'analyse de la vidéo (cf Chapitre 5.10) montre que le bridge fonctionne: le comportement du robot est similaire à celui sur Isaac.

5.9 Amélioration des performances

Les premiers essais affichent un facteur temps-réel[26] autour des 10 à 15%.

En utilisant le *profiler* de Gazebo [76], on peut capturer des intervalles de temps et les annoter, pour identifier ce qui ralenti les cycles de simulation.

```
GZ_PROFILE_BEGIN("Label");
...
GZ_PROFILE_END();
```

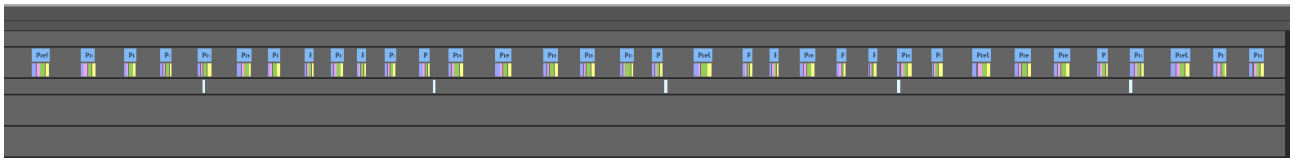


Fig. 19. – Profiling d'une simulation avec *gz-unitree*

Chaque groupe de segment correspond à un cycle de simulation.

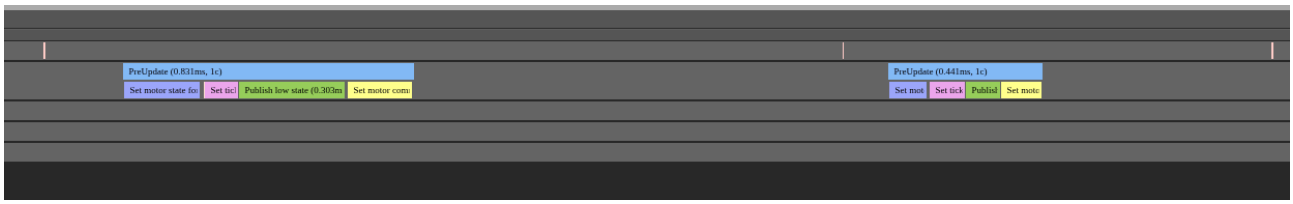
Prenons un cycle en particulier:

`::PreUpdate` 0.267 ms			
Update state 0.051 ms	Tick+CRC 0.039 ms	Publish state 0.142 ms	Update cmd. 0.028 ms

Tableau 3. – Profiling d'un cycle de simulation

Plus de la moitié du temps de calcul du plugin provient de l'envoi de l'état du robot sur le canal DDS `rt/lowstate`.

Notons également que, même si ce cyle-là a duré 0.267 ms, la durée d'un cycle est assez variable, certains atteignent 0.8 ms.



Quelques mesures ont été tentées pour réduire le temps nécéssaire à l'envoi d'un message DDS:

Restreindre DDS à `localhost` Il est possible que DDS envoie les messages en mode « broadcast », c'est-à-dire à

Déplacer dans un autre thread C'est ce qui a motivé la désynchronisation du thread « LowStateWriter » (cf Chapitre 5.6.2)

Ajuster la fréquence d'envoi Une fois `LowStateWriter` déplacé dans un thread indépendant, on peut ajuster la fréquence d'envoi, le thread étant récurrant[27]

Ainsi que d'autres optimisations, qui ne sont pas en rapport avec cette phase d'un cycle:

[26] Appelé RTF [75] (Real-Time Factor). Un RTF de 100% signifie que la simulation s'éxécute à vitesse réelle, un RTF inférieur à 1 signifie que la simulation est plus lente que la vitesse simulée

[27] Créé avec `CreateRecurrentThreadEx`

Mise en cache de joints à l'initialisation du plugin	pour éviter de devoir appeler `model.JointByName` dans une *hot loop*[28].
Utilisation d'une implémentation de CRC32 plus rapide	tentative avec *CRC++* [77] non achevée, à cause d'un *stack smashing* pendant l'éxécution

Après optimisations, on arrive à atteindre un RTF aux alentours des 30%. Des recherches supplémentaires sont nécéssaires pour atteindre un RTF raisonnable.

5.10 Enregistrement automatique de vidéos

Gazebo possède une fonctionnalité d'enregistrement vidéo, ce qui s'avère utile pour partager des résultats de simulation.

Cependant, l'enregistrement vidéo n'est pas nativement contrôlable par du code. L'idée était en effet de faire automatiquement tourner une simulation à chaque changement de la politique RL, et d'obtenir la vidéo du résultat, pour en observer l'évolution.

Il a donc fallu développer un autre plugin, héritant de `gz::gui::Plugin` cette fois-ci. Ce plugin écoute des messages sur des topics Gazebo, `/gui/record_video/...`, et permet de démarrer et arrêter l'enregistrement, tout en indiquant le chemin vers le fichier mp4 de sortie.

Au final, un script complet permettant de démarrer une simulation et l'enregistrer en MP4 ressemble à ceci

```
# Envoyer un message Gazebo avec un argument de type String et une valeur de retour
de type Booléen
send_to_gz() {
  gz service -s $1 --reqtype gz.msgs.StringMsg --reptype gz.msgs.Boolean --req "data:
\"$2\""
}

# Lancement en arrière plan
gz sim robot.sdf & sim_pid=$!
# On attends que la simulation soit prête
sleep 30

# Lancement de l'enregistrement
send_to_gz /gui/record_video/start mp4

# Lancement de la politique RL
uv run policy.py & policy_pid=$!
# On décide de la durée maximale de la vidéo (si la politique ne s'arrête pas d'elle
même)
sleep 120
kill $policy_pid

# Arrêt de l'enregistrement
send_to_gz /gui/record_video/stop file:///tmp/result.mp4

# Arrêt du simulateur
kill $sim_pid

# La vidéo est disponible à /tmp/result.mp4
```

[28] Boucle (`for` ou `while`) dont le corps est exécuté un très grand nombre de fois, et dont la rapidité est importante

5.11 Mise en CI/CD

On appelle CI/CD (pour *Continuous Integration / Continuous Delivery*) la pratique consistant à intégrer fréquemment des petits changements à un dépôt de code source commun, en lançant des tests régulièrement (partie « CI ») et éventuellement déployer la base de code fréquemment (partie « CD ») [78].

Une fois l'enregistrement vidéo rendu automatisable, si l'on veut mettre en place le lancement automatique à chaque commit du dépôt git de la politique (i.e. chaque changement de la politique), il faut créeer une description de *workflow* (dans notre cas, un workflow *Github Actions*).

Un workflow est un ensemble de commandes à exécuter dans un environnement virtualisé (qu'il s'agisse d'une machine virtuelle ou d'un simple container), ainsi que des évènements et conditions décrivant quand lancer l'éxécution (par exemple, « à chaque commit sur la branche `main` »). C'est un des outils permettant de mettre en place la CI/CD.

5.11.1 Une image de base avec Docker

L'environnement d'éxécution des workflows ne comporte pas d'installation de Gazebo. Étant donné le temps de compilation élevé, on peut « factoriser » cette étape dans une *image de base*, de laquelle on démarre pour chaque éxécution du workflow, dans laquelle tout les programmes nécéssaires sont déjà installés.

Pour cela, on part d'une image Ubuntu, dans lequelle on installe le nécéssaire: Just (pour lancer des commandes, un sorte de Makefile mais plus moderne [79]), FFMpeg (pour l'encodage H.264 servant à la création du fichier vidéo), XVFB (pour émuler un serveur X, cf Chapitre 5.11.2.2), Python (pour lancer la politique RL), et Gazebo.

```
FROM ubuntu:24.04

RUN apt update
RUN apt install -y curl just sudo
# Python (via le gestionnaire de versions et dépendances UV)
COPY --from=ghcr.io/astral-sh/uv:latest /uv /uvx /bin/

# Code source de gz-unitree
COPY . .

# Gazebo et outils de compilation
RUN just setup

# FFMpeg, XVFB
RUN apt install -y git ffmpeg xvfb xterm


# Compilation et installation de de gz-unitree
RUN mkdir -p /usr/local/lib/gz-unitree/
RUN just install
```

Un autre workflow, celui-là vivant dans le dépôt de gz-unitree, crée une image Docker depuis ce Dockerfile, qui est ensuite utilisable via `ghcr.io/Gepetto/gz-unitree` [80].

5.11.2 Une pipeline Github Actions

Une fois cette image disponible, on peut l'utiliser dans un workflow Github:

```
...
```

```yaml
jobs:
  test:
    runs-on: ubuntu-latest
    container:
      image: ghcr.io/gepetto/gz-unitree:latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v5
      - ...
```

Et lancer la simulation et l'enregistrement vidéo.

Pour récupérer le fichier vidéo final, on peut utiliser la notion d'*artifacts* de Github Actions:

```yaml
      - name: Save video as artifact
        uses: actions/upload-artifact@v4
        with:
          name: gz-unitree-video
          path: /tmp/result.mp4
```

Un environnement de développement contraignant
Développer et débugger une définition de workflow peut s'avérer complexe et particulièrement chronophage: n'ayant pas d'accès interactif au serveur éxécutant celui-ci, il faut envoyer ses changements au dépôt git, attendre que le workflow s'éxécute entièrement, et regardé si quelque chose s'est mal passé.

Par exemple, si jamais des fichiers sont manquants, ou ne sont pas au chemin attendu, il faut modifier le workflow pour y rajouter des instruction listant le contenu d'un répertoire (en utilisant `ls` ou `tree`, par exemple), lancer le workflow à nouveau et regarder les logs.

Ceci rend le développement assez fastidieux, surtout quand le workflow s'éxécute pendant des dizaines de minutes.

Émuler un serveur graphique

Les environnements de CI/CD s'apparentent plus à des serveurs qu'à des ordinateurs complets: en particulier, il n'y a pas d'interface graphique et donc pas de serveur d'affichage (*display server*).

Mais Gazebo nécéssite un display server pour enregistrer une vidéo.

Il convient donc de simuler un serveur d'affichage. Dans notre cas, l'environnement de CI/CD étant sous Linux, on simule un serveur X11 avec *XVFB* [81].

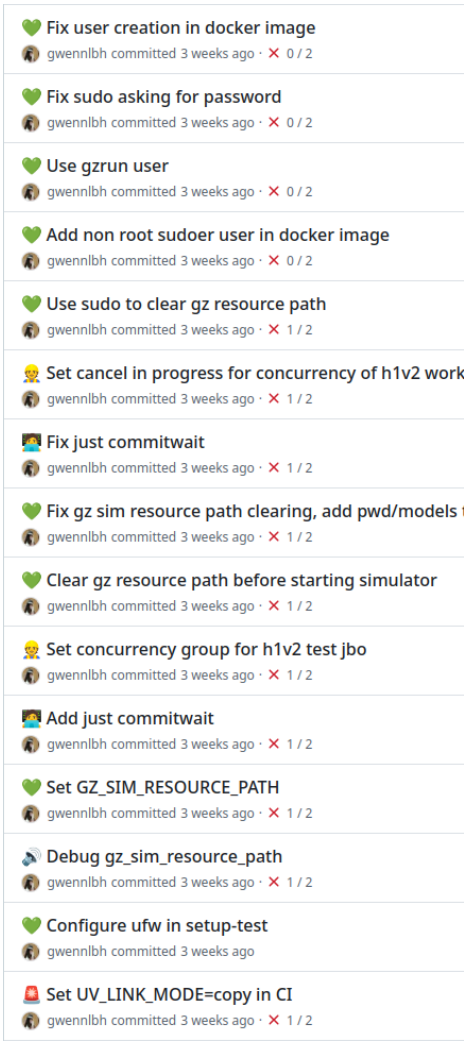


Fig. 20. – Quelques commits liés au développement du workflow[29]

[29]Les émojis servent d'icônes pour différencier les types de commits, via le standard Gitmoji [82]

6 Conclusion

Il est désormais possible d'utiliser le simulateur modulaire, open-source et communautaire *Gazebo* pour entraîner des politiques de reinforcement learning sur le robot *H1v2* de la société Unitree.

Bien que la reproductibilité de compilation ne soit pas encore atteinte, l'utilisation du gestionnaire de paquets Nix semble possible pour obtenir des garanties de reproductibilité.

Les performances du *bridge* SDK2 ⇆ Gazebo sont encore à améliorer, mais son utilisation est envisageable dans un context asynchrone, où le développement ne demande pas d'attendre les résultats de la simulation, via la pratique du CI/CD, par exemple.

Bibliographie

[1] Shengbo Eben Li, *Reinforcement Learning for Sequential Decision and Optimal Control*. Springer Singapore, p. 1-460. doi: 10.1007/978-981-19-7784-8.

[2] Tambet Matiisen, « Demystifying deep reinforcement learning », 19 décembre 2015, *Computational Neuroscience Research Group at University of Tartu*. Consulté le: 13 octobre 2025. [En ligne]. Disponible sur: https://web.archive.org/web/20180407053740/http://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/

[3] R. Sutton et A. Barto, *Reinforcement Learning: An Introduction*. in A Bradford book. MIT Press, 1998. Consulté le: 27 octobre 2025. [En ligne]. Disponible sur: https://books.google.fr/books?id=CAFR6IBF4xYC

[4] T. G. Dietterich, « Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition », *CoRR*, 1999, Consulté le: 2002. [En ligne]. Disponible sur: https://arxiv.org/abs/cs/9905014

[5] V. François-Lavet, R. Fonteneau, et D. Ernst, « How to Discount Deep Reinforcement Learning: Towards New Dynamic Strategies », *CoRR*, 2015, Consulté le: 13 octobre 2025. [En ligne]. Disponible sur: http://arxiv.org/abs/1512.02011

[6] Nick Bostrom, « Ethical Issues in Advanced Artificial Intelligence », 2003, *Int. Institute of Advanced Studies in Systems Research and Cybernetics*. Consulté le: 8 octobre 2025. [En ligne]. Disponible sur: https://nickbostrom.com/ethics/ai

[7] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, et P. Abbeel, « Trust Region Policy Optimization », févr. 2015, Consulté le: 13 octobre 2025. [En ligne]. Disponible sur: http://arxiv.org/abs/1502.05477v5

[8] « Trust Region Policy Optimization — Spinning Up documentation ». Consulté le: 14 octobre 2025. [En ligne]. Disponible sur: https://spinningup.openai.com/en/latest/algorithms/trpo.html#background

[9] David Pollard, *Asymptotia, Ch. 3, "Distances and affinities between measures"*. 2000, p. 6-7. Consulté le: 13 octobre 2025. [En ligne]. Disponible sur: https://web.archive.org/web/20150412031925/http://www.stat.yale.edu/~pollard/Books/Asymptopia/Metrics.pdf

[10] David J. C. MacKay, *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003, p. 34. Consulté le: 13 octobre 2025. [En ligne]. Disponible sur: https://books.google.fr/books?id=AKuMj4PN_EMC&lpg=PA34&pg=PA34#v=onepage&q&f=false

[11] Z. Xie, « Simple Policy Optimization », janv. 2024, Consulté le: 16 octobre 2025. [En ligne]. Disponible sur: http://arxiv.org/abs/2401.16025v2

[12] « Proximal Policy Optimization — Spinning Up documentation ». Consulté le: 16 octobre 2025. [En ligne]. Disponible sur: https://spinningup.openai.com/en/latest/algorithms/ppo.html

[13] M. Stępień, R. Kourdis, C. Roux, et O. Stasse, « Latent Conditioned Loco-Manipulation Using Motion Priors », septembre 2025. Consulté le: 31 octobre 2025. [En ligne]. Disponible sur: https://hal.science/hal-05242643

[14] NVIDIA Developer, « Isaac Sim - Robotics Simulation and Synthetic Data Generation ». Consulté le: 10 octobre 2025. [En ligne]. Disponible sur: https://developer.nvidia.com/isaac/sim

[15] NVIDIA Developer, « PhysX SDK - Latest Features & Libraries ». Consulté le: 10 octobre 2025. [En ligne]. Disponible sur: https://developer.nvidia.com/physx-sdk

[16] Consulté le: 16 juin 2025. [En ligne]. Disponible sur: https://mujoco.org/

[17] « MuJoCo simulate tutorial ». Consulté le: 10 octobre 2025. [En ligne]. Disponible sur: https://youtu.be/P83tKA1iz2Y

[18] Consulté le: 6 juin 2025. [En ligne]. Disponible sur: https://gazebosim.org/

[19] « Gazebo Sim: Physics engines ». Consulté le: 10 octobre 2025. [En ligne]. Disponible sur: https://gazebosim.org/api/sim/9/physics.html

[20] J. Lee *et al.*, « Dart: Dynamic animation and robotics toolkit », *The Journal of Open Source Software*, vol. 3, n° 22, p. 500, 2018.

[21] Bullet Physics SDK, *bullet3*. (12 avril 2011). GitHub. Consulté le: 10 octobre 2025. [En ligne]. Disponible sur: https://github.com/bulletphysics/bullet3

[22] « Bullet Real-Time Physics Simulation | Home of Bullet and PyBullet: physics simulation for games, visual effects, robotics and reinforcement learning. ». Consulté le: 10 octobre 2025. [En ligne]. Disponible sur: https://pybullet.org/wordpress/

[23] Erwin Coumans, « Bullet Physics Simulation Constraint Solving and Featherstone Articulated Body Algorithm ». International Conference and Exhibition on Computer Graphics and Interactive Technologies, 2015. Consulté le: 10 octobre 2025. [En ligne]. Disponible sur: https://docs.google.com/presentation/d/1wGUJ4neOhw5i4pQRfSGtZPE3CIm7MfmqfTp5aJKuFYM/edit?slide=id.g644a5aa5f_0_16#slide=id.g644a5aa5f_0_16

[24] Roy Featherstone, « Robot Dynamics Algorithms », 1978, *Springer New York, NY*.

[25] « Unitree H1 / H1-2 ». Consulté le: 30 juin 2025. [En ligne]. Disponible sur: https://www.unitree.com/h1/

[26] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, et D. Meger, « Deep Reinforcement Learning that Matters », sept. 2017, Consulté le: 16 octobre 2025. [En ligne]. Disponible sur: http://arxiv.org/abs/1709.06560v3

[27] Y. Gamage, D. Tiwari, M. Monperrus, et B. Baudry, « The Design Space of Lockfiles Across Package Managers », mai 2025, Consulté le: 17 octobre 2025. [En ligne]. Disponible sur: http://arxiv.org/abs/2505.04834v2

[28] P. Fernique, « AutoWIG : automatisation de l'encapsulation de librairies C++ en Python et en R », in *48èmes Journées de Statistique de la SFdS Montpellier*, Montpellier, France, mai 2016. Consulté le: 17 octobre 2025. [En ligne]. Disponible sur: https://inria.hal.science/hal-01316276

[29] Brian Lonsdorf, « Professor Frisby's Mostly Adequate Guide to Functional Programming », 2015, *Github*. Consulté le: 4 septembre 2025. [En ligne]. Disponible sur: https://github.com/MostlyAdequate/mostly-adequate-guide/blob/master/ch03.md

[30] « Reproducible Builds ». Consulté le: 4 septembre 2025. [En ligne]. Disponible sur: https://reproducible-builds.org/

[31] Fortran 2015 Committee Draft (J3/17-007r2), *ISO/IEC JTC 1/SC 22/WG5/N2137*. International Organization for Standardisation, 2017, p. 336-338. Consulté le: 4 septembre 2025. [En ligne]. Disponible sur: https://wg5-fortran.org/N2101-N2150/N2137.pdf

[32] « Relationship Between Routines, Functions, and Procedures », 13 janvier 2025, *IBM*. Consulté le: 4 septembre 2025. [En ligne]. Disponible sur: https://www.ibm.com/docs/en/informix-servers/15.0.0?topic=statement-relationship-between-routines-functions-procedures

[33] « Different Types of Robot Programming Languages », 2015, *Plant Automation Technology*. Consulté le: 4 septembre 2025. [En ligne]. Disponible sur: https://www.plantautomation-technology.com/articles/different-types-of-robot-programming-languages

[34] « Imperative programming: Overview of the oldest programming paradigm », 21 mai 2021, *IONOS*. Consulté le: 4 septembre 2025. [En ligne]. Disponible sur: https://www.ionos.com/digitalguide/websites/web-development/imperative-programming/

[35] Bill Hoffman et Kenneth Martin, *The Architecture of Open Source Applications (Volume 1) CMake*. Consulté le: 4 septembre 2025. [En ligne]. Disponible sur: https://aosabook.org/en/v1/cmake.html

[36] Consulté le: 19 mai 2025. [En ligne]. Disponible sur: https://nix.dev/manual/nix/2.17/language/

[37] Consulté le: 3 septembre 2025. [En ligne]. Disponible sur: https://search.nixos.org/packages

[38] NixOS Wiki Authors, « Nixpkgs/Contributing ». Consulté le: 3 septembre 2025. [En ligne]. Disponible sur: https://wiki.nixos.org/wiki/Nixpkgs/Contributing

[39] « Cachix — Nix binary cache hosting ». Consulté le: 3 septembre 2025. [En ligne]. Disponible sur: https://www.cachix.org/

[40] « Nix (package manager) — Sandboxing ». Consulté le: 3 septembre 2025. [En ligne]. Disponible sur: https://wiki.nixos.org/wiki/Nix_(package_manager)#Internals

[41] « GitHub-hosted runners », *Github*. Consulté le: 4 septembre 2025. [En ligne]. Disponible sur: https://docs.github.com/en/actions/concepts/runners/github-hosted-runners

[42] Consulté le: 6 juin 2025. [En ligne]. Disponible sur: https://www.raspberrypi.com/

[43] Fernando Borretti, « NixOS for the Impatient », 7 mai 2023. Consulté le: 4 septembre 2025. [En ligne]. Disponible sur: https://borretti.me/article/nixos-for-the-impatient

[44] *nix-ros-overlay*. (12 mars 2019). GitHub. Consulté le: 3 novembre 2025. [En ligne]. Disponible sur: https://github.com/lopsided98/nix-ros-overlay

[45] « [WIP] gz-sim and dependencies: init by ShamrockLee · Pull Request #394757 · NixOS/nixpkgs · GitHub ». Consulté le: 3 novembre 2025. [En ligne]. Disponible sur: https://github.com/NixOS/nixpkgs/pull/394757

[46] « ROS stuff: part 1 by Pandapip1 · Pull Request #355629 · NixOS/nixpkgs · GitHub ». Consulté le: 3 novembre 2025. [En ligne]. Disponible sur: https://github.com/NixOS/nixpkgs/pull/355629

[47] Gelakais, « Pull requests on NixOS/nixpkgs ». Consulté le: 3 novembre 2025. [En ligne]. Disponible sur: https://github.com/NixOS/nixpkgs/pulls?q=is%3Apr+gz+author%3AGuelakais

[48] Gepetto, *gazebros2nix*. (31 août 2025). GitHub. Consulté le: 3 novembre 2025. [En ligne]. Disponible sur: https://github.com/Gepetto/gazebros2nix

[49] « Eclipse Cyclone DDS - Home ». Consulté le: 17 octobre 2025. [En ligne]. Disponible sur: https://cyclonedds.io/

[50] Object Management Group, « DDS Interoperability Wire Protocol Specification Version 2.5 ». Consulté le: 24 juin 2025. [En ligne]. Disponible sur: https://www.omg.org/spec/DDSI-RTPS/

[51] Object Management Group, « Interface Definition Language Specification Version 4.2 ». Consulté le: 18 juin 2025. [En ligne]. Disponible sur: https://www.omg.org/spec/IDL/4.2/About-IDL

[52] Unitree, *unitree_sdk2/include/unitree/idl/hg/LowCmd_.hpp at main@2025-10-17, lines 33 to 63*. Github. Consulté le: 17 octobre 2025. [En ligne]. Disponible sur: https://github.com/unitreerobotics/unitree_sdk2/blob/main@%7B2025-10-17%7D/include/unitree/idl/hg/LowCmd_.hpp#L33-L63

[53] W.-Y. Liang, Y. Yuan, et H.-J. Lin, « A Performance Study on the Throughput and Latency of Zenoh, MQTT, Kafka, and DDS », mars 2023, Consulté le: 17 octobre 2025. [En ligne]. Disponible sur: http://arxiv.org/abs/2303.09419v1

[54] Unitree, *unitreerobotics/unitree_sdk2, main@2025-10-17*. Github. Consulté le: 17 octobre 2025. [En ligne]. Disponible sur: https://github.com/unitreerobotics/unitree_sdk2/tree/main@%7B2025-10-17%7D

[55] « IMPORTED_LOCATION — CMake 4.2.0-rc1 Documentation ». Consulté le: 17 octobre 2025. [En ligne]. Disponible sur: https://cmake.org/cmake/help/latest/prop_tgt/IMPORTED_LOCATION.html

[56] « Ghidra - Powerful Open-Source Reverse Engineering Tool ». Consulté le: 17 octobre 2025. [En ligne]. Disponible sur: https://ghidralite.com/

[57] Unitree Robotics, *unitree_mujoco*. (1 novembre 2021). GitHub. Consulté le: 17 octobre 2025. [En ligne]. Disponible sur: https://github.com/unitreerobotics/unitree_mujoco

[58] « High-Level Motion Services Interface - □□□□ □□□□ ». Unitree. Consulté le: 30 octobre 2025. [En ligne]. Disponible sur: https://support.unitree.com/home/en/H1_developer/Sports_Services_Interface

[59] « HelloWorld source code — Eclipse Cyclone DDS, 0.11.0 ». Consulté le: 22 octobre 2025. [En ligne]. Disponible sur: https://cyclonedds.io/docs/cyclonedds/latest/getting_started/helloworld/helloworld_source_code.html

[60] Unitree Robotics, *unitree_ros2*. (20 octobre 2023). GitHub. Consulté le: 22 octobre 2025. [En ligne]. Disponible sur: https://github.com/unitreerobotics/unitree_ros2

[61] « Protocols/rtps - Wireshark Wiki ». Consulté le: 22 octobre 2025. [En ligne]. Disponible sur: https://wiki.wireshark.org/Protocols/rtps

[62] « SDFormat Specification, world.plugin element ». Consulté le: 22 octobre 2025. [En ligne]. Disponible sur: http://sdformat.org/spec?ver=1.12&elem=world#world_plugin

[63] « Gazebo Sim: Finding resources ». Consulté le: 22 octobre 2025. [En ligne]. Disponible sur: https://gazebosim.org/api/sim/8/resources.html#:~:text=All%20paths%20on%20the%20GZ_SIM_SYSTEM_PLUGIN_PATH%20environment%20variable

[64] « SDFormat Specification, world.plugin.filename element ». Consulté le: 22 octobre 2025. [En ligne]. Disponible sur: http://sdformat.org/spec?ver=1.12&elem=world#plugin_filename

[65] « Gazebo Transport: Nodes and Topics ». Gazebo. Consulté le: 29 octobre 2025. [En ligne]. Disponible sur: https://gazebosim.org/api/transport/8/nodestopics.html

[66] « Protocol Buffers Documentation ». Consulté le: 29 octobre 2025. [En ligne]. Disponible sur: https://protobuf.dev/

[67] « Gazebo Transport: Messages ». Gazebo. Consulté le: 29 octobre 2025. [En ligne]. Disponible sur: https://gazebosim.org/api/transport/8/messages.html

[68] cplusplus.com, « std::bind ». Consulté le: 23 octobre 2025. [En ligne]. Disponible sur: https://cplusplus.com/reference/functional/bind/

[69] L. T. H. Gam, D. H. Quan, P. V. B. Ngoc, B. H. Quan, et B. T. Thanh, « Position–Force Control of a Lower-Limb Rehabilitation Robot Using a Force Feed-Forward and Compensative Gravity Proportional Derivative Method », *Electronics*, vol. 13, nº 22, 2024, doi: 10.3390/electronics13224494.

[70] « rt/lowState topic Descirption - □□□□ □□□□ ». Unitree. Consulté le: 23 octobre 2025. [En ligne]. Disponible sur: https://support.unitree.com/home/en/H1_developer/H1-2_Basic_Services_Interface#heading-4

[71] M. Araki, « PID Control », *Control Systems, Robotics, and Automation*, vol. 2. 2009. Consulté le: 23 octobre 2025. [En ligne]. Disponible sur: http://www.eolss.net/ebooks/Sample%20Chapters/C18/E6-43-03-03.pdf

[72] « rt/lowCmd topic Description - □□□□ □□□□ ». Unitree. Consulté le: 23 octobre 2025. [En ligne]. Disponible sur: https://support.unitree.com/home/en/H1_developer/H1-2_Basic_Services_Interface#heading-1

[73] Unitree Robotics, *unitree_sdk2/example/h1/low_level/motors.hpp at 211dad48a40fb79485ad3d34c0ec6f6b12b0d8a5*. Github. Consulté le: 29 octobre 2025. [En ligne]. Disponible sur: https://github.com/unitreerobotics/unitree_sdk2/blob/211dad48a40fb79485ad3d34c0ec6f6b12b0d8a5/example/h1/low_level/motors.hpp#L54-L75

[74] Unitree Robotics, *unitree_mujoco/example/cpp/stand_go2.cpp at 24b7a498f6bc032684a8c8af30876a84d53d03e4*. Github. Consulté le: 29 octobre 2025. [En ligne]. Disponible sur: https://github.com/unitreerobotics/unitree_mujoco/blob/24b7a498f6bc032684a8c8af30876a84d53d03e4/example/cpp/stand_go2.cpp#L54-L84

[75] Centre for Tactile Internet with Human-in-the-Loop, « Simulation Speed in Gazebo ». Consulté le: 28 octobre 2025. [En ligne]. Disponible sur: https://ceti.pages.st.inf.tu-dresden.de/robotics/howtos/SimulationSpeed.html

[76] « Gazebo Common: Profiler ». Consulté le: 28 octobre 2025. [En ligne]. Disponible sur: https://gazebosim.org/api/common/7/profiler.html

[77] Daniel Bahr, *CRCpp*. (1 mai 2016). GitHub. Consulté le: 29 octobre 2025. [En ligne]. Disponible sur: https://github.com/d-bahr/CRCpp

[78] I. Sacolick, « What is CI/CD? Continuous integration and continuous delivery explained », 1 juin 2021, *InfoWorld*. Consulté le: 29 octobre 2025. [En ligne]. Disponible sur: https://www.infoworld.com/article/2269266/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html

[79] « Introduction - Just Programmer's Manual ». Consulté le: 28 octobre 2025. [En ligne]. Disponible sur: https://just.systems/man/en/

[80] Gepetto, *Package gz-unitree*. Github. Consulté le: 28 octobre 2025. [En ligne]. Disponible sur: https://github.com/Gepetto/gz-unitree/pkgs/container/gz-unitree

[81] I. David P. Wiggins The Open Group, « XVFB », *X.org*. Consulté le: 4 septembre 2025. [En ligne]. Disponible sur: https://www.x.org/archive/X11R7.7/doc/man/man1/Xvfb.1.xhtml

[82] Carlos Cuesta, « gitmoji | An emoji guide for your commit messages ». Consulté le: 29 octobre 2025. [En ligne]. Disponible sur: https://gitmoji.dev/

A Preuves

A.1 Cas dégénéré de $D_{\mathrm{KL}}(Q, Q') = 0$ sans utilisation de max

Soit S (resp. $A \subset \mathbb{N}$) l'espace des états (resp. actions) de l'environnement. Soit $Q : S \times A \to [0, 1]$ une distribution de probabilité du choix par l'agent d'une action dans un état tel que

$$\forall s \in S, Q(s, 1) = Q(s, 2) \tag{27}$$

Soit $Q' : S \times A \to [0, 1]$ définit ainsi:

$$\forall s \in S, Q'(s, 1) \coloneqq 2Q(s, 1) \tag{28}$$

$$\forall s \in S, Q'(s, 2) \coloneqq \frac{1}{2}Q(s, 2) \tag{29}$$

$$\forall s \in S, \forall a \in A - \{1, 2\}, Q'(s, a) \coloneqq Q(s, a) \tag{30}$$

On a

$$\begin{aligned}
D_{\mathrm{KL}}(Q \parallel Q') &= \sum_{(s,a)\in S\times A} Q(s, a) \log \frac{Q(s, a)}{Q'(s, a)} \\
&\text{On découpe la somme selon les valeurs de } A\text{:} \\
&= \sum_{s\in S} \sum_{a\in A-\{1,2\}} \left[Q(s, a) \log \frac{Q(s, a)}{Q'(s, a)} \right] + Q(s, 1) \log \frac{Q(s, 1)}{Q'(s, 1)} + Q(s, 2) \log \frac{Q(s, 2)}{Q'(s, 2)} \\
&= \sum_{s\in S} \underbrace{\sum_{a\in A-\{1,2\}} \cancel{Q(s, a) \log \frac{Q(s, a)}{Q(s, a)}}}_{\text{d'après (30)}} + Q(s, 1) \log \underbrace{\frac{Q(s, 1)}{2Q(s, 1)}}_{\text{d'après (28)}} + Q(s, 2) \log \underbrace{\frac{Q(s, 2)}{\frac{1}{2}Q(s, 2)}}_{\text{d'après (29)}} \\
&= \sum_{s\in S} Q(s, 1) \Big[\log Q(s, 1) - \log Q(s, 1) - \log 2 \Big] + \\
&\qquad Q(s, 2) \Big[\log Q(s, 2) - \log Q(s, 2) - \log \frac{1}{2} \Big] \\
&= \sum_{s\in S} -Q(s, 1) \log 2 + Q(s, 2) \log 2 \\
&= \sum_{s\in S} \log 2 \underbrace{\cancel{(Q(s, 2) - Q(s, 1))}}_{\text{d'après (27)}} \\
&= \sum_{s\in S} 0 = 0
\end{aligned} \tag{31}$$

A.2 $\eta(\pi, r)$ comme une espérance

Soit r une fonction récompense et π une politique. Soit C une variable aléatoire à valeurs dans $\mathcal{S}$, dont la loi de probabilité suit celle de π.

On a

$$\begin{aligned}\exp\left(\sum_{t=0}^{\infty}\gamma^t r(C_t)\right) &= \sum_{(c_t)_{t\in\mathbb{N}}\in\mathcal{S}}\left(\sum_{t=0}^{\infty}\gamma^t r(c_t)\right)\mathbb{P}\left(\sum_{t=0}^{\infty}\gamma^t r(C_t)=\sum_{t=0}^{\infty}\gamma^t r(c_t)\right)\\ &= \sum_{(c_t)_{t\in\mathbb{N}}\in\mathcal{S}}\left(\sum_{t=0}^{\infty}\gamma^t r(c_t)\right)\mathbb{P}\Big(C=(c_t)_{t\in\mathbb{N}}\Big)\end{aligned} \tag{32}$$

Soit S (resp. A) la suite des premiers (resp. deuxièmes) éléments de C, c'est-à-dire $\forall t\in\mathbb{N}, (S_t, A_t) \coloneqq C_t$.

Étant donné la définition de $\mathcal{S}$:

- S_t dépend de A_{t-1} et S_{t-1}
- A_t dépend de S_t

On a alors, pour toute suite $(c_t)_{t\in\mathbb{N}} \in \mathcal{S}$:

$$\begin{gathered}P\Big(C=(c_t)_{t\in\mathbb{N}}\Big) = \\ \mathbb{P}(S_0=s_0)\mathbb{P}(A_0=a_0 \mid S_0=s_0)\prod_{t=1}^{\infty}\mathbb{P}(S_t=s_t \mid C_{t-1}=c_{t-1})\mathbb{P}(A_t=a_t \mid S_t=s_t)\end{gathered} \tag{33}$$

On a

$$\begin{aligned}\mathbb{P}(S_0=s_0) &= \rho_0(s_0)\\ \forall t\in\mathbb{N},\quad \mathbb{P}(A_t=a_t \mid S_t=s_t) &= Q_\pi(s_t,a_t)\\ \forall t\in\mathbb{N}^*,\quad \mathbb{P}(S_t=s_t \mid C_{t-1}=c_{t-1}) &= \mathbb{P}(M(C_{t-1})=M(c_{t-1}) \mid C_{t-1}=c_{t-1})\\ &= \mathbb{P}(C_{t-1}=c_{t-1} \mid C_{t-1}=c_{t-1}) = 1\end{aligned} \tag{34}$$

Donc on a

$$\begin{aligned}P\Big(C=(c_t)_{t\in\mathbb{N}}\Big) &= \rho_0(s_0)Q_\pi(s_0,a_0)\prod_{t=1}^{\infty}Q_\pi(s_t,a_t)\\ &= \rho_0(s_0)\prod_{t=0}^{\infty}Q_\pi(s_t,a_t)\end{aligned} \tag{35}$$

Et ainsi

$$\begin{aligned}\exp\left(\sum_{t=0}^{\infty}\gamma^t r(C_t)\right) &= \sum_{(c_t)_{t\in\mathbb{N}}\in\mathcal{S}}\left(\sum_{t=0}^{\infty}\gamma^t r(c_t)\right)\mathbb{P}\Big(C=(c_t)_{t\in\mathbb{N}}\Big)\\ &= \sum_{(c_t)_{t\in\mathbb{N}}\in\mathcal{S}}\left(\sum_{t=0}^{\infty}\gamma^t r(c_t)\right)\rho_0(s_0)\prod_{t=0}^{\infty}Q_\pi(s_t,a_t)\\ &= \eta(\pi,r)\quad\blacksquare\end{aligned} \tag{36}$$

A.3 Simplification de l'expression de $L(s, a, \Pi, \Pi', R)$ dans PPO-Clip

Soit $(s, a) \in S \times A$, et Π' une politique. Posons $\alpha \coloneqq A_{\Pi',R}(s, a)$, $q/q' \coloneqq Q_\Pi(s, a)/Q_{\Pi'}(s, a)$.

Cas $\alpha > 0$	**Cas $\alpha < 0$**
$L(s, a, \Pi, \Pi', R)$	$L(s, a, \Pi, \Pi', R)$
$= \min\left(\frac{q}{q'}\alpha, \quad \text{clip}\left(\frac{q}{q'}, 1-\varepsilon, 1+\varepsilon\right)\alpha\right)$	$= \min\left(\frac{q}{q'}\alpha, \quad \text{clip}\left(\frac{q}{q'}, 1-\varepsilon, 1+\varepsilon\right)\alpha\right)$
$= \min\left(\frac{q}{q'}, \quad \text{clip}\left(\frac{q}{q'}, 1-\varepsilon, 1+\varepsilon\right)\right)\alpha$	$= \max\left(\frac{q}{q'}, \quad \text{clip}\left(\frac{q}{q'}, 1-\varepsilon, 1+\varepsilon\right)\right)\alpha$
car $\alpha > 0$	car $\alpha < 0$

...et $q/q' \in [1-\varepsilon, 1+\varepsilon]$

$= \min\left(\frac{q}{q'}, \quad \text{clip}\left(\frac{q}{q'}, 1-\varepsilon, 1+\varepsilon\right)\right)\alpha$	$= \max\left(\frac{q}{q'}, \quad \text{clip}\left(\frac{q}{q'}, 1-\varepsilon, 1+\varepsilon\right)\right)\alpha$
$= \min\left(\frac{q}{q'}, \quad \frac{q}{q'}\right)\alpha$	$= \max\left(\frac{q}{q'}, \quad \frac{q}{q'}\right)\alpha$
$= \min\left(\frac{q}{q'}, 1+\varepsilon\right)\alpha$	$= \max\left(\frac{q}{q'}, 1-\varepsilon\right)\alpha$
car $1+\varepsilon > \frac{q}{q'}$	car $1-\varepsilon < \frac{q}{q'}$

...et $q/q' > 1+\varepsilon$

$= \min\left(\frac{q}{q'}, \quad \text{clip}\left(\frac{q}{q'}, 1-\varepsilon, 1+\varepsilon\right)\right)\alpha$	$= \max\left(\frac{q}{q'}, \quad \text{clip}\left(\frac{q}{q'}, 1-\varepsilon, 1+\varepsilon\right)\right)\alpha$
$= \min\left(\frac{q}{q'}, \quad 1+\varepsilon\right)\alpha$	$= \max\left(\frac{q}{q'}, \quad 1+\varepsilon\right)\alpha$
	$= \max\left(\frac{q}{q'}, \quad 1-\varepsilon\right)\alpha$
	car $1-\varepsilon < 1+\varepsilon < \frac{q}{q'}$

...et $q/q' < 1-\varepsilon$

$= \min\left(\frac{q}{q'}, \quad \text{clip}\left(\frac{q}{q'}, 1-\varepsilon, 1+\varepsilon\right)\right)\alpha$	$= \max\left(\frac{q}{q'}, \quad \text{clip}\left(\frac{q}{q'}, 1-\varepsilon, 1+\varepsilon\right)\right)\alpha$
$= \min\left(\frac{q}{q'}, \quad 1-\varepsilon\right)\alpha$	$= \max\left(\frac{q}{q'}, \quad 1-\varepsilon\right)\alpha$
$= \min\left(\frac{q}{q'}, \quad 1+\varepsilon\right)\alpha$	
car $1+\varepsilon > 1-\varepsilon > \frac{q}{q'}$	

B Implémentation de CRC32

Extraite du code source de *unitree_mujoco*, sous license BSD-3-Clause [74]

```
uint32_t crc32_core(uint32_t *ptr, uint32_t len)
{
    unsigned int xbit = 0;
    unsigned int data = 0;
    unsigned int CRC32 = 0xFFFFFFFF;
    const unsigned int dwPolynomial = 0x04c11db7;

    for (unsigned int i = 0; i < len; i++)
    {
        xbit = 1 << 31;
        data = ptr[i];
        for (unsigned int bits = 0; bits < 32; bits++)
        {
            if (CRC32 & 0x80000000)
            {
                CRC32 <<= 1;
                CRC32 ^= dwPolynomial;
            }
            else
            {
                CRC32 <<= 1;
            }

            if (data & xbit)
                CRC32 ^= dwPolynomial;
            xbit >>= 1;
        }
    }

    return CRC32;
}
```