



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

**Gwenn Le Bihan**

gwenn.lebihan7@gmail.com

ENSEEIH

22 Novembre 2025

# 1 Remerciements

## Table des matières

1	Remerciements .....	2
2	Contexte .....	3
2.1	Bases théoriques du <i>Reinforcement Learning</i> .....	3
2.1.1	L'entraînement .....	4
2.1.2	Deep Reinforcement Learning .....	6
2.1.3	Tendances à la « tricherie » des agents .....	6
2.1.3.1	Sous-spécification de la fonction coût .....	6
2.1.3.2	Bug dans l'implémentation de l'environnement .....	6
2.1.3.3	La validation comme méthode de mitigation .....	6
2.2	Entraînement par <i>curriculum</i> .....	7
2.3	Mise à jour de la politique .....	7
2.3.1	<i>Q-learning</i> .....	7
2.3.2	Évaluation de la performance d'une politique .....	7
2.3.2.1	Chemins d'états possibles $\mathcal{C}$ .....	8
2.3.2.2	Récompense attendue $\eta$ .....	9
2.3.2.3	Avantage $A$ .....	10
2.3.2.4	Lien entre $\eta$ et $A$ .....	10
2.3.2.5	<i>Surrogate advantage</i> $\mathcal{L}$ .....	11
2.3.3	<i>Trust Region Policy Optimization</i> .....	11
2.3.3.1	Distance entre politiques .....	12
2.3.3.2	Pourquoi faire le maximum sur chaque $s \in S$ ? .....	12
2.3.3.3	Région de confiance .....	13
2.3.4	<i>Proximal Policy Optimization</i> .....	13
2.3.4.1	Avec pénalité ( <i>PPO-Penalty</i> ) .....	13
2.3.4.2	Par <i>clipping</i> ( <i>PPO-Clip</i> ) .....	13
2.4	Application en robotique .....	14
2.4.1	Spécification de la tâche .....	14
2.4.1.1	Définition explicite de la fonction coût .....	14
2.4.1.2	Apprentissage par des exemples .....	14
2.4.2	Inventaire des simulateurs en robotique .....	14
2.4.2.1	Isaac .....	14
2.4.2.2	MuJoCo .....	15
2.4.2.3	Gazebo .....	15
2.4.3	Inventaire des moteurs de simulation physique .....	15
2.4.3.1	DART .....	15
2.4.3.2	Bullet .....	15
2.4.3.3	Bullet avec Featherstone .....	15
2.5	Le <i>H1v2</i> d'Unitree .....	15
2.6	Reproductibilité logicielle .....	15
3	Packaging reproductible avec Nix .....	17
3.1	Reproductibilité .....	17
3.1.1	État dans le domaine de la programmation .....	17

3.1.2	Contenir les effets de bords .....	17
3.1.3	État dans le domaine de la robotique .....	17
3.1.4	Environnements de développement .....	18
3.2	Nix, le gestionnaire de paquets pur .....	18
3.2.1	Un <i>DSL</i> <sup>1</sup> fonctionnel .....	18
3.2.2	Un écosystème de dépendances .....	19
3.2.3	Une compilation dans un environnement fixé .....	20
3.2.3.1	Un complément utile: compiler en CI .....	20
3.3	NixOS, un système d'exploitation à configuration déclarative .....	20
3.4	Packaging Nix pour <i>gz-unitree</i> .....	20
4	Étude du SDK d'Unitree et du bridge SDK $\Leftarrow$ MuJoCo .....	20
4.1	Canaux DDS .....	21
4.2	Une base de code partiellement open-source .....	22
4.3	Un autre bridge existant: <code>unitree_mujoco</code> .....	23
5	Développement du bridge SDK $\Leftarrow$ Gazebo .....	25
5.1	Établissement du contact .....	25
5.2	Installation du plugin dans Gazebo .....	26
5.3	Architecture du plugin .....	27
5.4	Réception des commandes .....	28
5.5	Émission de l'état .....	29
5.6	Désynchronisations .....	31
5.7	Essai sur des politiques réelles .....	32
5.8	Amélioration des performances .....	32
5.9	Enregistrement de vidéos .....	32
5.9.1	Contrôle programmatique de l'enregistrement .....	32
5.10	Mise en CI/CD .....	32
5.10.1	Une image de base avec Docker .....	32
5.10.2	Une pipeline Github Actions .....	32
	Bibliographie .....	32
A	Preuves .....	36
A.1	Cas dégénéré de $D_{\text{KL}}(Q, Q') = 0$ sans utilisation de max .....	36
A.2	$\eta(p, r)$ comme une espérance .....	36
A.3	Simplification de l'expression de $L(s, a, \mathcal{P}, \mathcal{P}', R)$ dans PPO-Clip .....	38

## 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:

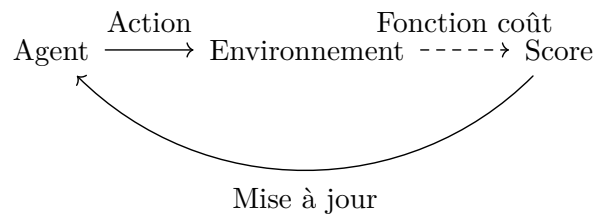
- Un *agent*: c'est le programme que l'on souhaite créer
- Des *actions* que l'agent peut choisir d'effectuer ou pas
- Un *environnement*, que les actions viennent modifier

---

<sup>1</sup>Domain-Specific Language

- 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 défini comme « qui minimise le coût » (ou maximise la récompense):



Cette technique est particulièrement adaptée aux 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, on a des correspondances claires pour ces quatre notions:

<b>Agent</b>	Robot pour lequel on développe le programme de contrôle (appelée une <i>politique</i> )
<b>Actions</b>	Envoi d'ordres aux moteurs
<b>Environnement</b>	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
<b>Coût</b>	un ensemble de contraintes (« ne pas endommager le robot »), dont la plupart dépendent de l'objectif de la politique

### 2.1.1 L'entraînement

(TODO: Expliquer exploration vs exploitation et  $\gamma$ )

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. – Exemple d'agent à mémoire exhaustive 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'entière 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*:

$$L : E \rightarrow S \quad (1)$$

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<sup>2</sup> 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  x - 2 $
(0, C'est plus)	+1	$L(x + 1, ) = 2$ $L(x - 1, ) = 2$
(1, C'est plus)	+1	$L(x + 1, ) = 1$ $L(x - 1, ) = 2$
(3, C'est moins)	-1	$L(x + 1, ) = 2$ $L(x - 1, ) = 3$
(4, C'est moins)	-1	$L(x + 1, ) = 3$ $L(x - 1, ) = 4$
(5, C'est moins)	-1	$L(x + 1, ) = 4$ $L(x - 1, ) = 5$

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

---

<sup>2</sup>Reinforcement Learning

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

Cependant, ces ensembles sont bien souvent prohibitivement grands (e.g.  $x \in \llbracket 0, 10^{34} \rrbracket$ ), 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:

<b>État actuel</b>	devient la couche d'entrée
<b>Meilleure action</b>	devient la couche de sortie
<b>Coûts associés</b>	deviennent les neurones des couches cachées
<b>Le remplissage du tableau</b>	devient la rétropropagation pendant l'entraînement

### 2.1.3 Tendances à la « tricherie » des agents

Expérimentalement, on sait que des tendances « tricheuses » émergent facilement pendant l'entraînement [Réf. nécessaire]: 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 suffisamment bien en compte toutes les possibilités de l'environnement (autrement dit, il manque de contraintes).

#### Sous-spécification de la fonction coût

(Note: Bof cette partie )

Un exemple populaire est l'expérience de pensée du Maximiseur de trombones [2]: 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

#### Bug dans l'implémentation de l'environnement

Bien évidemment, pour l'agent, tant qu'un bug n'est pas explicitement découragé par sa prise en compte dans la fonction coût. Si une action est favorable à l'amélioration du score, l'agent la prendra.

#### La validation comme méthode de mitigation (Note: ça se dit mitigation en français?)

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 en fait 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 Entraînement par *curriculum*

## 2.3 Mise à jour de la politique

### 2.3.1 *Q-learning*

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

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

L'expression comporte deux hyperparamètres:

**Learning rate  $\alpha$**       contrôle à quel point l'on favorise l'évolution de  $Q$  ou pas.

**Discount factor  $\gamma$**       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].

### 2.3.2 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:

$A$	l'ensemble des actions
$S$	l'ensemble des états possibles de l'environnement
$\rho_0 : S \rightarrow [0, 1]$	la distribution de probabilité de l'état initial de l'environnement. Si l'on initialise l'environnement de manière uniformément aléatoire, $\rho_0$ est une équiprobabilité <sup>3</sup>
$M : S \times A \rightarrow S$	le moteur de simulation physique, qui applique l'action à un état de l'environnement et envoie le nouvel état de l'environnement
$\mathcal{P} : S \rightarrow A$	une politique
$\mathcal{P}^* : S \rightarrow A$	la meilleure politique possible, celle que l'on cherche à approcher

---

<sup>3</sup>i.e.  $\text{card } \rho_0(S) = 1$

- $R : S \rightarrow \mathbb{R}^+$  sa fonction de récompense
- $Q_p : S \times A \rightarrow [0, 1]$  sa distribution de probabilité, qu'on suppose Markovienne (elle ne dépend que de l'état dans lequel on est).  $Q_p(s_t, a_t)$  est la probabilité que  $p$  choisisse  $a_t$  quand on est dans l'état  $s_t$  ( $s_t$  est l'état **pré**-action, et non post-action)
- $Q$  et  $Q^*$   $Q_{\mathcal{P}}$  et  $Q_{\mathcal{P}^*}$ , pour alléger les notations

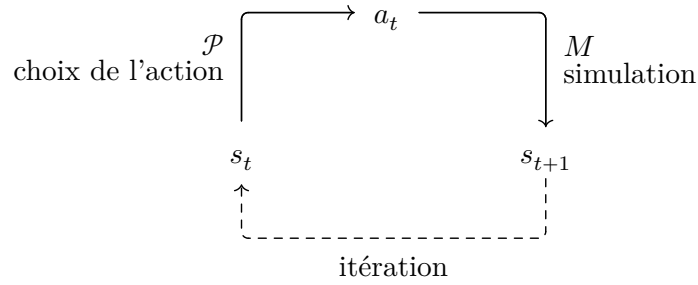
On suppose  $A$  et  $S$  dénombrables<sup>4</sup>.

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:

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

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

$M$  et  $\mathcal{P}$  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  $\mathcal{P}$  à jour à chaque tour de boucle) ou l'utiliser:



Quand on « déroule »  $\mathcal{P}$  en en partant d'un certain état initial  $s_0$ , on obtient une suite d'états et d'actions:

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

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

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

<sup>4</sup>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



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

(Note: p-ê Expliquer pourquoi une suite de  $S$  en fait ça marche pas, en gros on choppe pas tt les chemins possible psk faut trouver  $a$  en fonction de  $p$  donc ya pas tout. Si on prend  $p(a)$  c'est que le chemin que la politique prendrait)

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

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

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

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

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, peu importe l'action choisie.

On note aussi que  $\mathcal{C}$  (et donc  $\mathcal{C}_p$  aussi) est dénombrable, étant construit à partir de  $(S \times A)^{\mathbb{N}}$  et  $S$ ,  $A$  et  $\mathbb{N}$  étant aussi dénombrables<sup>5</sup>

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

(Note: pas sûre de cette phrase)

### Récompense attendue $\eta$

$\eta$  représente la récompense moyenne à laquelle l'on peut s'attendre pour une politique  $p$  avec fonction de récompense  $r$ .

Elle prend en compte le *discount factor*  $\gamma$  : les récompenses des actions deviennent de moins en moins<sup>6</sup> importantes avec le temps.  $\eta$  est définie ainsi [6]

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

On peut également exprimer  $\eta(p, r)$  comme une espérance. Soit  $C$  une variable aléatoire de  $\mathcal{S}$ . On a (cf preuve en A.2)

<sup>5</sup>On a  $\text{card } \mathcal{C} \leq \text{card}((S \times A)^{\mathbb{N}}) = \text{card}(S \times A)^{\text{card } \mathbb{N}} = (\text{card } S \text{ card } A)^{\text{card } \mathbb{N}} \leq (\aleph_0)^{\text{card } \mathbb{N}} = 2^{\aleph_0} = \aleph_0$

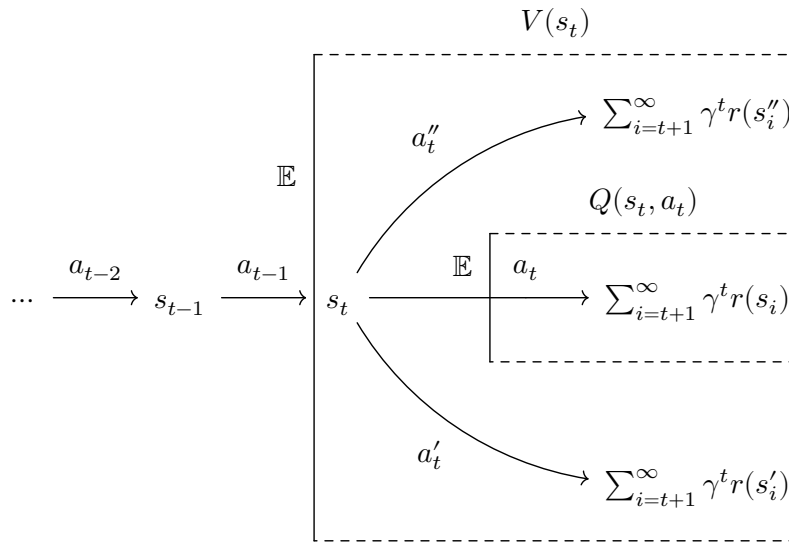
<sup>6</sup>En supposant  $\gamma < 1$ , ce qui est souvent le cas [Réf. nécessaire] (TODO: Mettre dans la def de  $\gamma$ )

$$\eta(p, r) = \mathbb{E} \left( \sum_{t=0}^{\infty} \gamma^t r(C_t) \right) \quad (8)$$

### Avantage $A$

L'avantage  $A_{p,r}(s, a)$  mesure à quel point il est préférable de choisir l'action  $a$  quand on est dans l'état  $s$  (pour la politique  $p$ , avec « préférable » au sens de  $(r(S), \geq)$ )

On peut visualiser ce calcul ainsi:



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

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

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.

### Lien entre $\eta$ et $A$

Pour une fonction de récompense  $r$  donnée,  $A$  permet de calculer  $\eta$  pour une politique  $p'$  en fonction de la valeur de  $\eta$  pour une autre politique  $p'$  [7]

$$\begin{aligned} \eta(p', r) &= \eta(p, r) + \mathbb{E}_{(c_t)_{t \in \mathbb{N}} \sim p' \in \mathcal{S}} \sum_{t=0}^{\infty} \gamma^t A_{p,r}(c_t) \\ &\text{Qui se simplifie en [6]} \\ &= \eta(p, r) + \sum \end{aligned} \tag{10}$$

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

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

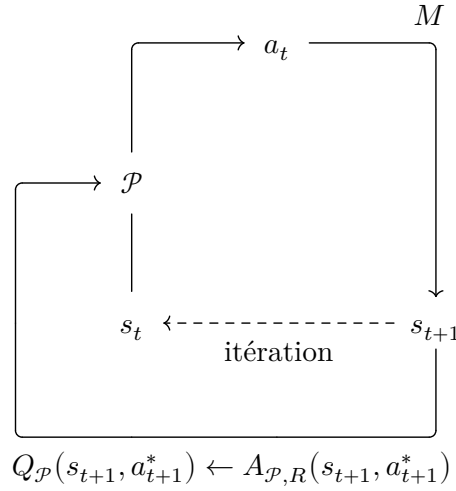


Fig. 5. – Boucle d'entraînement

Avec

$$a_{t+1}^* := \operatorname{argmax}_{a \in A} A_{\mathcal{P},R}(s_{t+1}, a) \tag{11}$$

Mais, en pratique, des erreurs d'approximations peuvent rendre  $A_{\mathcal{P},R}(s_{t+1}, a_{t+1}^*)$  négatif, ce qui empêche de s'en servir pour définir une valeur de  $Q_{\mathcal{P}}$  [6]

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

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

### **2.3.3 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.

$$Q' = \begin{cases} \operatorname{argmax}_q \mathcal{L}_r(q, Q) \\ \text{s.c. distance}(Q', Q) < \delta \end{cases} \quad (13)$$

Avec  $\delta$  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]:

$$D_{\text{KL}}(P \parallel P') := \sum_{x \in \mathcal{X}} P(x) \log \frac{P(x)}{P'(x)} \quad (14)$$

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  $Q_{\mathcal{P}}$  et  $Q_{\mathcal{P}'}$  pour  $s \in S$  fixé [6]

$$\max_{s \in S} D_{\text{KL}}(Q_{\mathcal{P}'}(s, \cdot) \parallel Q_{\mathcal{P}}(s, \cdot)) < \delta \quad (15)$$

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

### 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*.

(Note: C'est ma théorie ça, faudrait être sûr que le papier ne donne pas d'explications)

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

$$\forall s \in S, Q(s, 1) = Q(s, 2) \quad (16)$$

et

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

On a  $D_{\text{KL}}(Q, Q') = 0$  (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 tous 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  $\mathcal{P}'$  acceptables comme nouvelle politique, aussi appelé une *trust region* (région de confiance), d'où la méthode d'optimisation tire son nom [6].

En pratique, l'optimisation sous cette contrainte est trop demandeuse en puissance de calcul, on utilise plutôt l'espérance [6]

$$\overline{D_{\text{KL}}} := \mathbb{E}_{s \in S} D_{\text{KL}}(Q(s, \cdot) \parallel Q'(s, \cdot)) \quad (18)$$

### 2.3.4 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]

$$\begin{aligned} \arg\max_{\mathcal{P}'} \quad & \mathbb{E}_{(s,a) \in \mathcal{S}} L(s, a, \mathcal{P}, \mathcal{P}', R) \\ \text{s.c. } \quad & \top \end{aligned} \quad (19)$$

### Avec pénalité (*PPO-Penalty*)

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

$$L(s, a, \mathcal{P}, \mathcal{P}', R) = \frac{Q_{\mathcal{P}}(s, a)}{Q_{\mathcal{P}'}(s, a)} A_{\mathcal{P}, R}(s, a) - \beta D_{\text{KL}}(\mathcal{P} \parallel \mathcal{P}') \quad (20)$$

Avec  $\beta$  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]

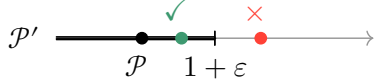
$$\begin{aligned} L(s, a, \mathcal{P}, \mathcal{P}', R) = \min \left( \frac{Q_{\mathcal{P}'}(s, a)}{Q_{\mathcal{P}}(s, a)} A_{\mathcal{P}', R}(s, a), \right. \\ \left. \text{clip} \left( \frac{Q_{\mathcal{P}'}(s, a)}{Q_{\mathcal{P}}(s, a)}, 1 - \varepsilon, 1 + \varepsilon \right) A_{\mathcal{P}', R}(s, a) \right) \end{aligned} \quad (21)$$

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

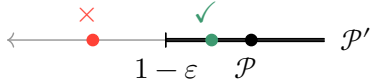
$$\text{clip} := (x, m, M) \mapsto \begin{cases} m & \text{si } x < m \\ M & \text{si } x > M \\ x & \text{sinon} \end{cases} \quad (22)$$

La complexité de l'expression, et la présence d'un min au lieu de simplement un clip est due au fait que l'avantage  $A_{\mathcal{P}',R}(s,a)$  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  $\mathcal{P}(s)$ .

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


**Si l'avantage est négatif** choisir  $a$  est pire que garder  $\mathcal{P}(s)$ .

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


## 2.4 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.

On est donc dans un cas où il est très utile de

Un environnement de RL<sup>7</sup> 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), 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*.

### 2.4.1 Spécification de la tâche

#### Définition explicite de la fonction coût

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:

- Puissance maximale sur les commandes envoyées aux moteurs
- (TODO: )

#### Apprentissage par des exemples

(TODO: Déterminer si je parle de ça, en fonction de cmb de pages il reste après avoir fait le reste, ça fera ptet trop...)

Il est possible d'éviter la définition manuelle de la fonction coût, ce qui requiert d'instrumentaliser l'environnement avec des capteurs supplémentaires, en fournissant à la place

### 2.4.2 Inventaire des simulateurs en robotique

#### Isaac

Un simulateur développé par NVIDIA [13], utilisant son propre moteur de rendu, PhysX [14]

<sup>7</sup>Reinforcement Learning

## MuJoCo

Un simulateur initialement propriétaire. Il a été rendu gratuit puis open source par Google DeepMind [15].

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 [16].

## Gazebo

Les intérêts de Gazebo [17] 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 [18], à l'inverse de MuJoCo.

Gazebo possède des plugins officiels pour:

<b>DART</b>	Plugin <code>gz-physics-dartsim-plugin</code> , c'est l'implémentation principale, et celle par défaut [18].
<b>Bullet</b>	Plugin <code>gz-physics-bulletsim-plugin</code> . En beta [18].
<b>Bullet Featherstone</b>	Plugin <code>gz-physics-bullet-featherstone-plugin</code> , également en beta [18].

## 2.4.3 Inventaire des moteurs de simulation physique

### DART

DART, pour Dynamic Animation and Robotics Toolkit [19],

### Bullet

Bullet [20], [21]

### Bullet avec Featherstone

L'algorithme de Featherstone [22], servant d'implémentation alternative à Bullet [23]

## 2.5 Le *H1v2* d'Unitree

Le *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) [24]

## 2.6 Reproductibilité logicielle

La reproductibilité est particulièrement complexe dans le champ du reinforcement learning [25].

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.

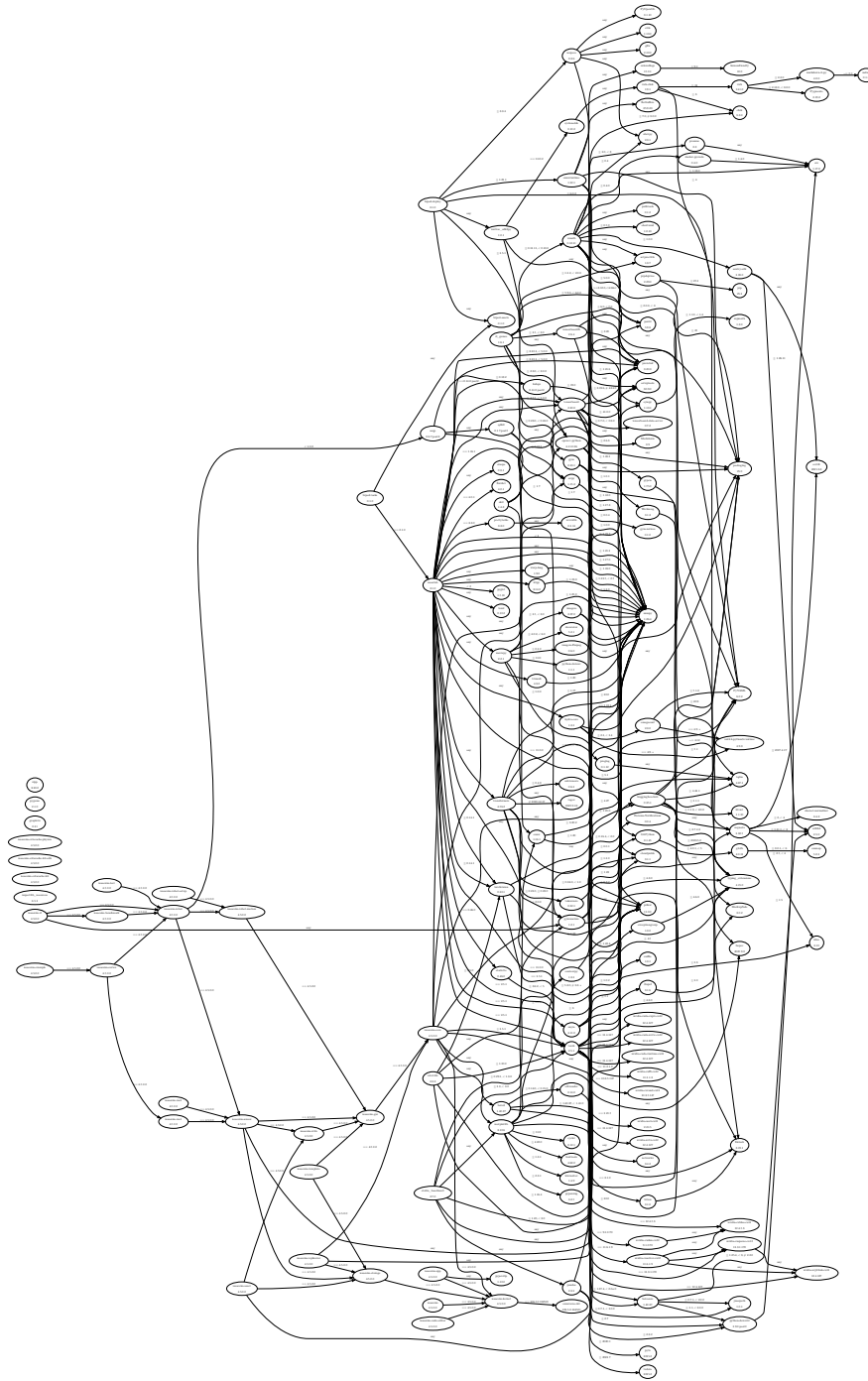


Fig. 8. – Arbre des dépendances pour *Gepetto/h1v2-Isaac*

Bien que toutes ces dépendances puissent être spécifiées à des versions strictes [26] 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 [27]. 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 praticité, on permet aux **fonctions** 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 [28].

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:

$$\forall E, F, \forall f : E \rightarrow F, \forall (e_1, e_2) \in E^2, e_1 = e_2 \Rightarrow f(e_1) = f(e_2) \quad (23)$$

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

```
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 [29]

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*<sup>8</sup> et une fonction classique [30]. Certaines fonctions, plutôt appelées *procédures*, sont uniquement composées d’effet de bord puisqu’elle ne renvoie pas de valeur [31]

#### 3.1.3 État dans le domaine de la robotique

En robotique, pour donner des ordres au matériel, on interagit 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 abstraitionnelle au matériel.

---

<sup>8</sup>sans effets de bord

<sup>9</sup>Il arrive assez communément d’utiliser Python, un langage haut-niveau, mais c’est dans ce cas à but de prototypage, et le code contrôlant les moteurs est écrit dans un langage bas niveau puis appelé par Python par FFI.

De fait, les langages employés sont communément C, C++ ou Python<sup>9</sup> [32], des langages bien plus impératifs que fonctionnels [33].

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++ [34])

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.

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

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 exécutables

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<sup>10</sup> 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

<pre>def g(x, y):     if y == 5:         x = 6     else:         x = 8     return f(x)</pre>	<pre>let g x y = f (     if y = 5 then         6     else         8 )</pre>
<b>Python</b> ( <i>if</i> et <i>else</i> sont des instructions)	<b>OCaml</b> ( <i>if</i> et <i>else</i> 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 [35]. Un fichier *.nix* définit une fonction, que Nix sait exécuter pour compiler le code source.

Expression d'une fonction en Python	En Nix
<code>lambda f(a): a + 3</code>	<code>{ a }: a + 3</code>

<sup>10</sup>Domain-Specific Language

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

```
{
  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 incluses ou reliées au binaire final)

### 3.2.2 Un écosystème 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* [36].

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* [37]

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 [38]

### 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 [39]

### Un complément utile: compiler en CI

Pour aller plus loin, on peut lancer la compilation du paquet Nix en *CI*<sup>12</sup>, 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 [40].

## 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 [41], 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'exécution 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` [42].

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*

(TODO: Faire cette partie)

## 4 Étude du SDK d'Unitree et du bridge SDK $\leftrightarrow$ MuJoCo

Unitree met à disposition du public un *SDK*<sup>13</sup> permettant de contrôler ses robots (dont le H1v2).

<sup>12</sup>Continuous Integration, lit. intégration continue

<sup>13</sup>Kit de développement logiciel (Software Development Kit)

## 4.1 Canaux DDS

Pour communiquer avec le robot via le réseau, Unitree utilise CycloneDDS, une implémentation par Oracle du standard DDS<sup>14</sup> [43], une technologie de communication bidirectionnelle<sup>15</sup> en temps réel, standardisée par l'Object Management Group, OMG [44]. 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 [45].

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++ [46]

DDS groupe les messages 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  $1\text{ GB s}^{-1}$ , pour des messages d'environ 1 kB chacun [47]. 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 *domains*.

<sup>14</sup>pour Data Distribution Service

<sup>15</sup>dite « *pub-sub* » pour *publish/subscribe*

## 4.2 Une base de code partiellement open-source

Le code source du SDK d'unitree est disponible sur Github [48]. 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 dépôt git

Compiler le SDK nécessite l'existence 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

```

Fig. 9. – 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 [49].

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/ddsc
├─ 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 planer quelques doutes sur la nature open-source du code: ces binaires requis sont-ils seulement présent pour améliorer l'expérience développeur en accélérant 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* [50].

Cependant, la découverte de l'existence d'un bridge officiel SDK  $\leftrightarrow$  Mujoco [51] a rendu cette piste non nécessaire.

### 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.

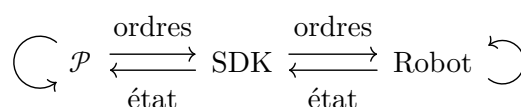


Fig. 10. – Fonctionnement usuel du SDK

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).

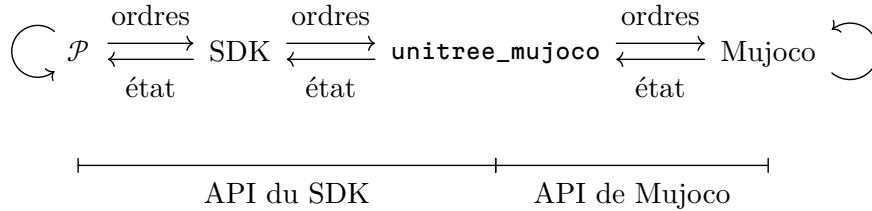


Fig. 11. – Fonctionnement via `unitree_mujoco` du SDK

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.

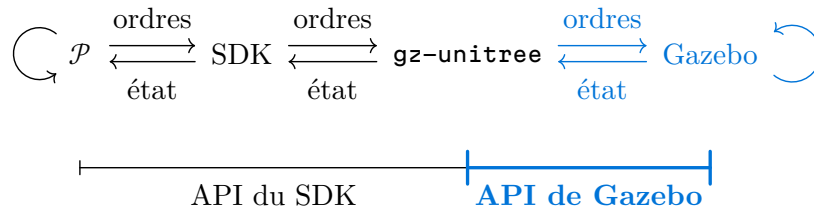


Fig. 12. – Fonctionnement via `gz-unitree` du SDK

Le bridge de Mujoco fonctionne en interceptant les messages sur le canal `rt/lowcmd` et 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, et non pas à des messages plus avancés tel que « avancer de  $x$  mètres » (TODO: ces messages plus haut-niveau = sport mode non? dire quand ils servent)

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)*<sup>16</sup> aux messages d'un canal (pour les recevoir)

<sup>16</sup>abonnement



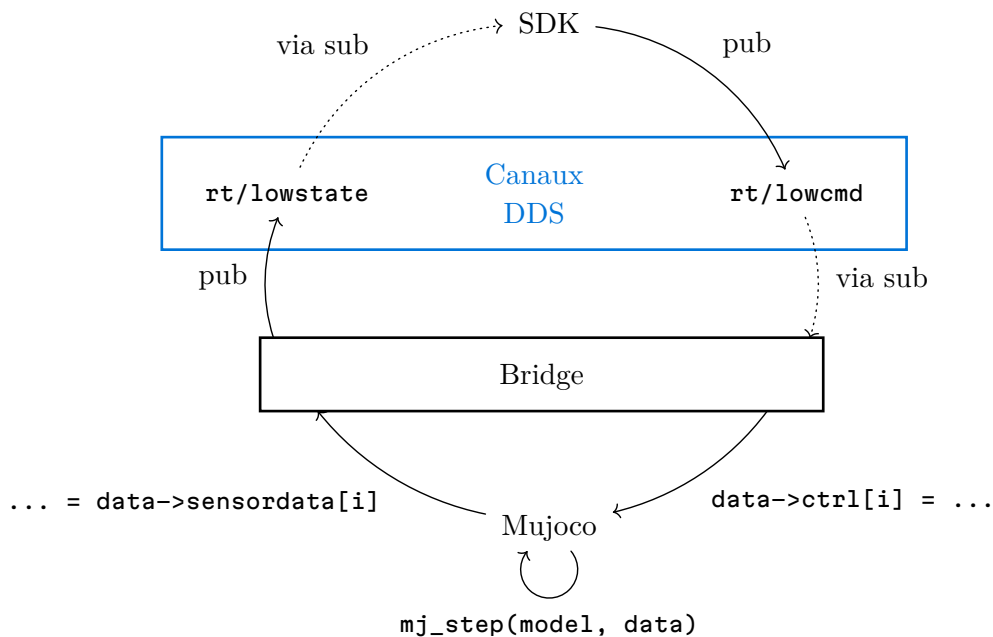


Fig. 13. – Cycle de vie de la simulation avec le bridge pour Mujoco

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 $\leftrightarrow$ 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 [52] `rt/lowcmd`, en récupérant les définitions IDL des messages, disponibles sur le dépôt `unitree_ros2`<sup>17</sup> [53]

On commence par importer la bibliothèque DDS et les définitions IDL de `rt/lowcmd`

```
#include "messages/LowCmd.hpp"
#include "dds/dds.h"
...

int main (int argc, char ** argv)
{
```

On initialise les différents objets permettant de lire sur un canal

```
dds_entity_t participant, topic, reader;
LowCmd_ *msg;
void *samples[MAX_SAMPLES];
...

participant = dds_create_participant(DDS_DOMAIN_DEFAULT, NULL, NULL);

topic = dds_create_topic(participant, &LowCmd__desc, "HelloWorldData_Msg",
NULL, NULL);
```

<sup>17</sup>`unitree_mujoco` n'avait pas encore été découvert

```

qos = dds_create_qos();
dds_qoset_reliability(qos, DDS_RELIABILITY_RELIABLE, DDS_SECS (10));

reader = dds_create_reader(participant, topic, qos, NULL);

dds_delete_qos(qos);

samples[0] = LowCmd__alloc();

```

Et on attend qu'un message arrive sur le canal, pour l'afficher

```

/* Poll until data has been read. */
while (true)
{
    rc = dds_read(reader, samples, infos, MAX_SAMPLES, MAX_SAMPLES);

    /* Check if we read some data and it is valid. */
    if ((rc > 0) && (infos[0].valid_data))
    {
        /* Print Message. */
        msg = (LowCmd_*) samples[0];
        printf("== [Subscriber] Received : ");
        fflush(stdout);
        break;
    }
    else
    {
        dds_sleepfor(DDS_MSECS(20));
    }
}

```

Enfin, on libère les ressources avant la terminaison du programme

```

LowCmd__free(samples[0], DDS_FREE_ALL);
dds_delete(participant);
return EXIT_SUCCESS;
}

```

Malheureusement, cette solution s'est avérée infructueuse, à cause de (ce qui sera compris bien plus tard) un problème de numéro de domaine DDS.

On change d'approche en préférant plutôt utiliser les abstractions fournies par le SDK de Unitree (cf Chapitre 5.4 et Chapitre 5.5)

Enfin, si un pare-feu est actif, il faut autoriser le trafic udp l'intervalle d'adresses 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

```

## 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 référant dans le fichier SDF [54], 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` [55], [56].

### 5.3 Architecture du plugin

Le plugin consiste en trois parties distinctes:

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

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

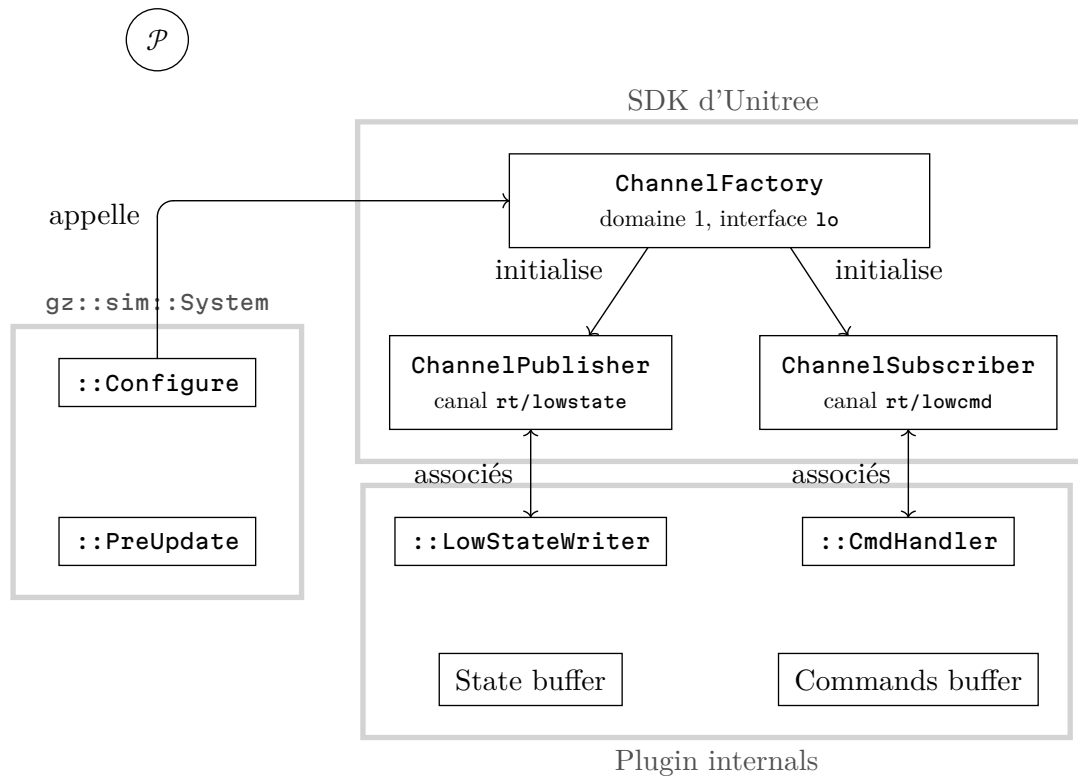


Fig. 14. – Phase d'initialisation du plugin

On commence par instancier un contrôleur dans le domaine DDS n°1, sur l'interface réseau **1o**<sup>18</sup>  
On lui associe:

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

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

## 5.4 Réception des commandes

Lorsqu'un message, publié par  $\mathcal{P}$  (1A) et contenant des ordres pour les moteurs, arrive sur **rt/lowcmd**, **::CmdHandler** est appelé (2, 3), et modifie un *buffer* (4) contenant la dernière commande reçue.

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.

<sup>18</sup>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

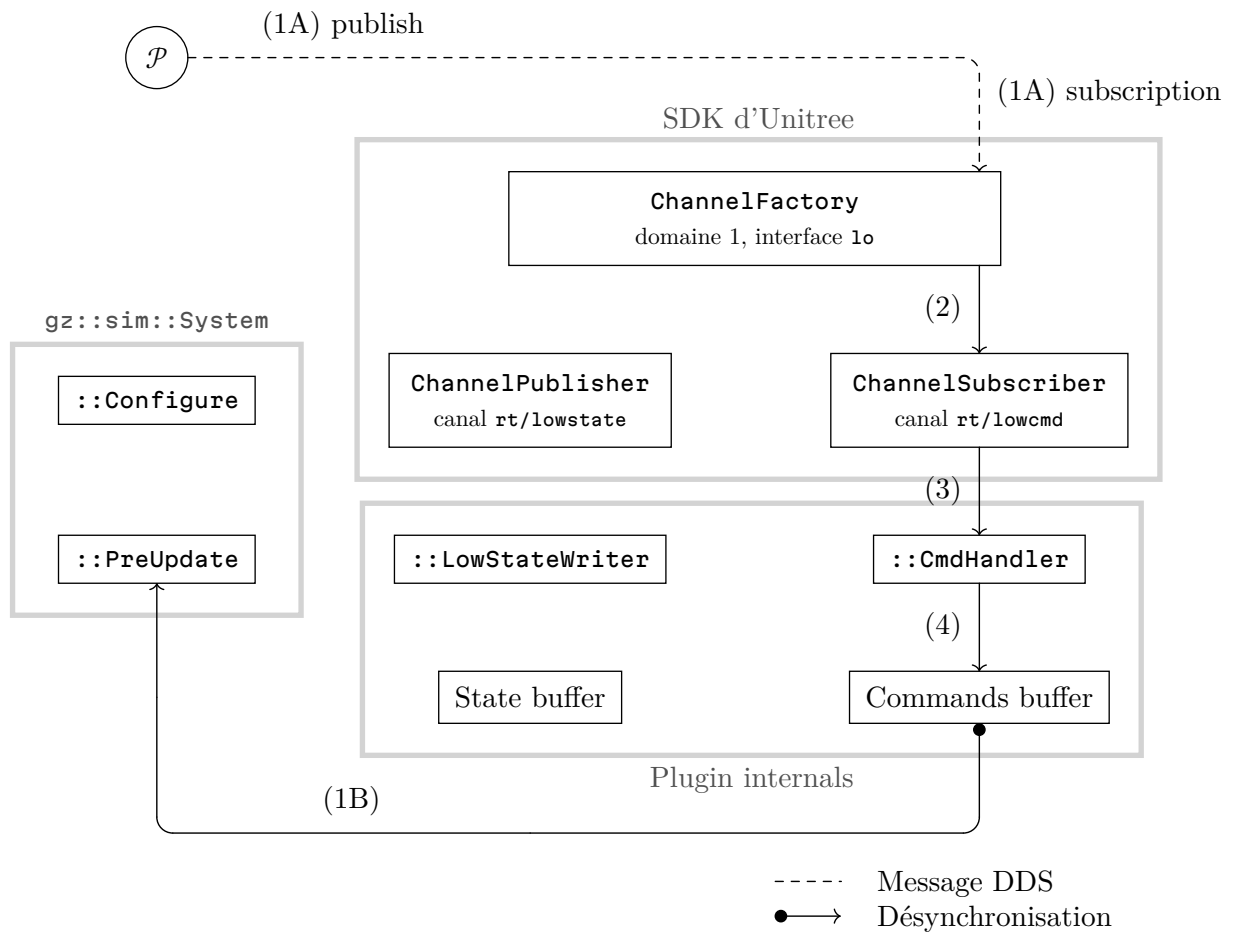


Fig. 15. – Phase de réception des commandes

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.

## 5.5 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).

Le `LowStateWriter` vient lire le *State buffer* (1B) pour publier l'état sur le canal DDS (2, 3) qui est ensuite lu par  $\mathcal{P}$  (4), qui (on le suppose) posséde une subscription sur `rt/lowstate`

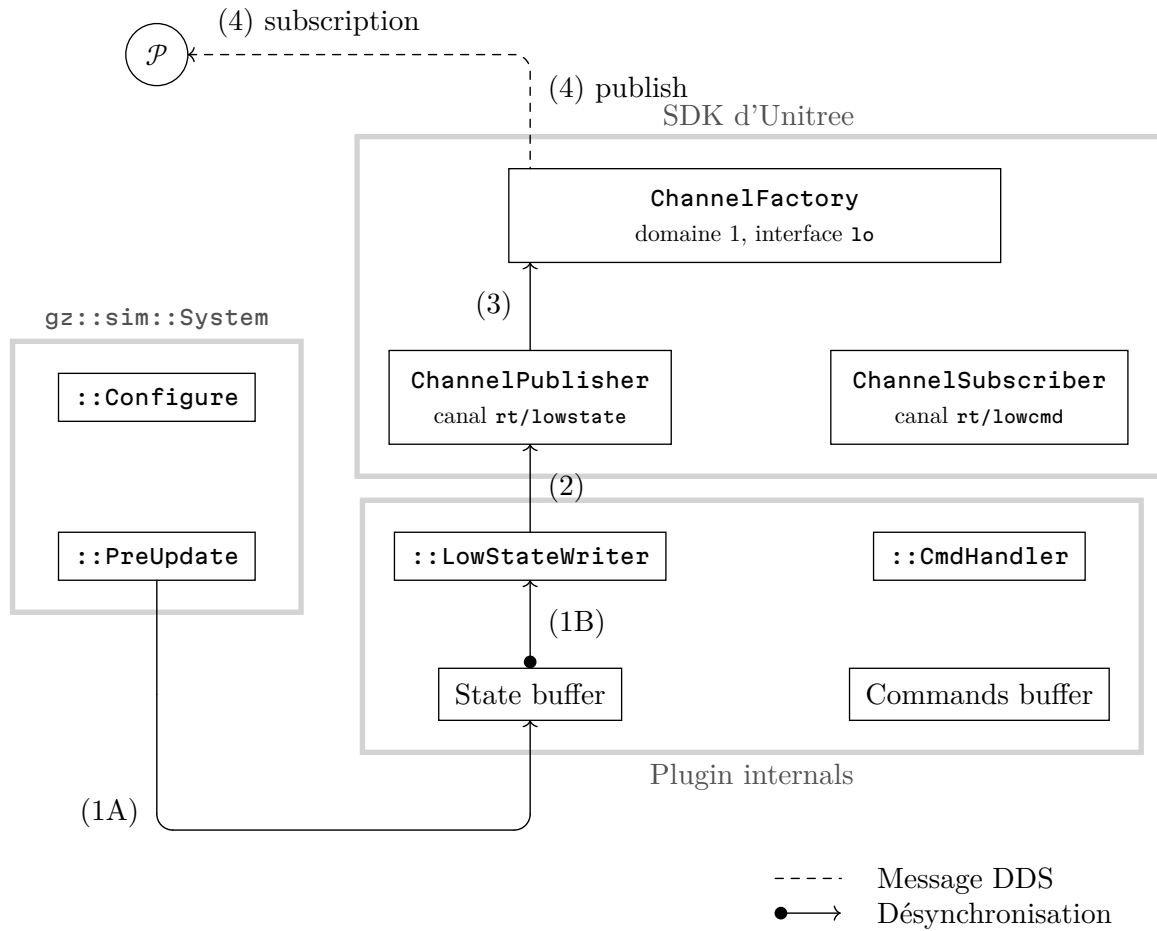


Fig. 16. – Phase d'envoi de l'état

Ici également, `LowStateWriter` s'exécute *en parallèle* du code de `::PreUpdate`: En effet, la création du `ChannelPublisher` démarre une boucle qui vient exécuter `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 *trois* boucles indépendantes qui sont en jeu:

- 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  $\mathcal{P}$  (à quelle fréquence  $\mathcal{P}$  est-elle capable de recevoir des messages)

Similairement à la réception de commandes:

**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<sup>19</sup>)  $\mathcal{P}$  sera moins grande

<sup>19</sup>En fonction de si `::LowStateWriter` est plus fréquente que  $\mathcal{P}$  (dans ce cas là, c'est ce qui est acceptable par  $\mathcal{P}$  qui est limitant) ou inversement (dans ce cas, c'est ce que la boucle du publisher met à disposition de  $\mathcal{P}$  qui est limitant)

**Si `::PreUpdate` est moins fréquente**  $\mathcal{P}$  recevra plusieurs fois le même état, ce qui sera représentatif du fait que la simulation n'a pas encore avancé.

## 5.6 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 trois boucles indépendantes, représentées ci-après:

- Celle de la simulation (en bleu), qui doit englober l'entièreté d'un cycle
- Celle du `ChannelPublisher` (en rouge)
- Celle de  $\mathcal{P}$  (en rose)

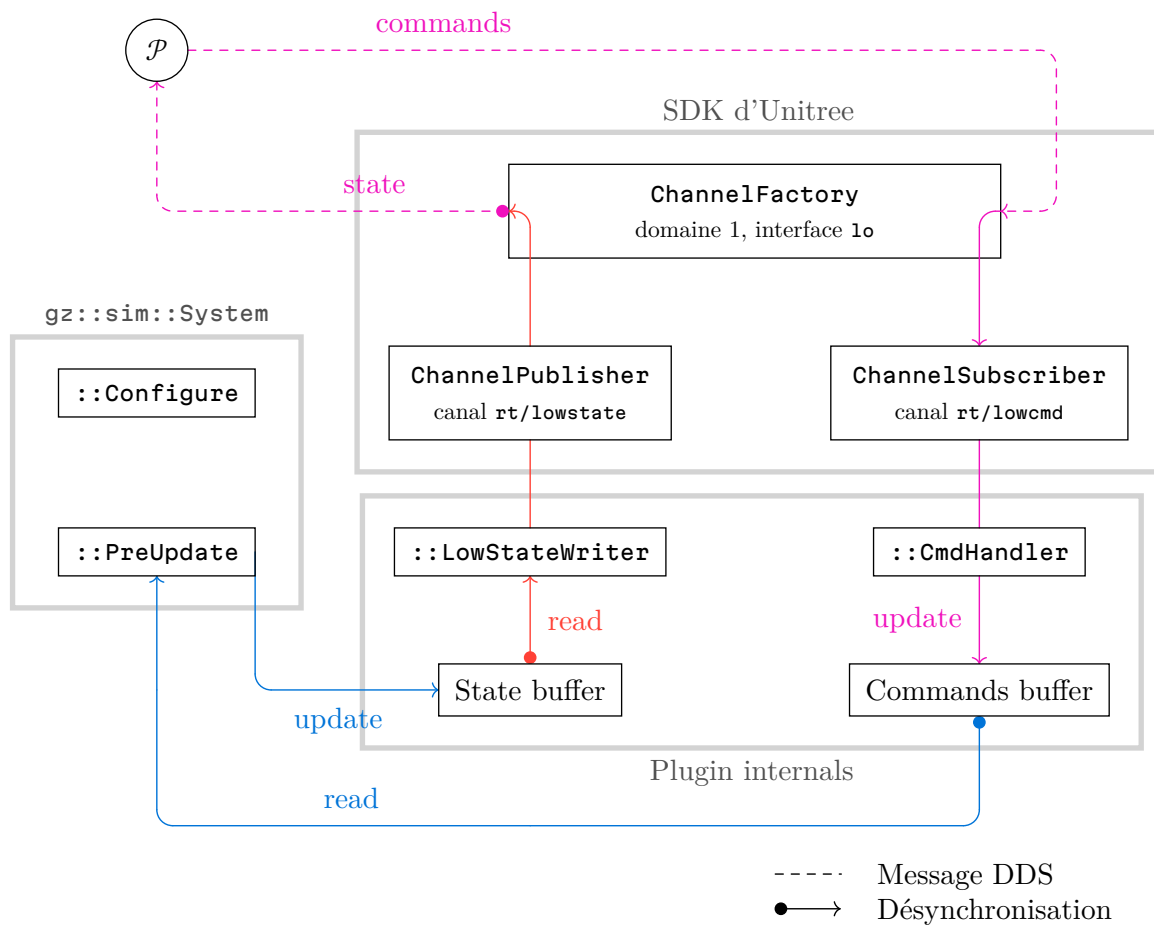


Fig. 17. – Cycle complet. Un cycle commence avec la flèche « update » partant de `::PreUpdate`

Ces désynchronisations pourraient expliquer les problèmes de performance rencontrés (cf Chapitre 5.8)

## 5.7 Essai sur des politiques réelles

## 5.8 Amélioration des performances

## 5.9 Enregistrement de vidéos

### 5.9.1 Contrôle programmatique de l'enregistrement

## 5.10 Mise en CI/CD

### 5.10.1 Une image de base avec Docker

### 5.10.2 Une pipeline Github Actions

## 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](https://doi.org/10.1007/978-981-19-7784-8).
- [2] 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>
- [3] 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/>
- [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] 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>
- [7] J. Langford, « Approximately Optimal Approximate Reinforcement Learning », p. 267-274, 2002.
- [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](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] 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>
- [14] NVIDIA Developer, « PhysX SDK - Latest Features & Libraries ». Consulté le: 10 octobre 2025. [En ligne]. Disponible sur: <https://developer.nvidia.com/physx-sdk>
- [15] Consulté le: 16 juin 2025. [En ligne]. Disponible sur: <https://mujoco.org/>
- [16] « MuJoCo simulate tutorial ». Consulté le: 10 octobre 2025. [En ligne]. Disponible sur: <https://youtu.be/P83tKA1iz2Y>
- [17] Consulté le: 6 juin 2025. [En ligne]. Disponible sur: <https://gazebo-sim.org/>
- [18] « Gazebo Sim: Physics engines ». Consulté le: 10 octobre 2025. [En ligne]. Disponible sur: <https://gazebo-sim.org/api/sim/9/physics.html>
- [19] J. Lee *et al.*, « Dart: Dynamic animation and robotics toolkit », *The Journal of Open Source Software*, vol. 3, n° 22, p. 500, 2018.
- [20] Bullet Physics SDK, *bullet3*. (12 avril 2011). GitHub. Consulté le: 10 octobre 2025. [En ligne]. Disponible sur: <https://github.com/bulletphysics/bullet3>
- [21] « 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/>
- [22] Roy Featherstone, « Robot Dynamics Algorithms », 1978, *Springer New York, NY*.
- [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/1wGUJ4neOhw5i4pQRfSGtZPE3CIIm7MfmqfTp5aJKuFYM/edit?slide=id.g644a5aa5f\\_0\\_16#slide=id.g644a5aa5f\\_0\\_16](https://docs.google.com/presentation/d/1wGUJ4neOhw5i4pQRfSGtZPE3CIIm7MfmqfTp5aJKuFYM/edit?slide=id.g644a5aa5f_0_16#slide=id.g644a5aa5f_0_16)
- [24] « Unitree H1 / H1-2 ». Consulté le: 30 juin 2025. [En ligne]. Disponible sur: <https://www.unitree.com/h1/>
- [25] 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>

- [26] 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>
- [27] P. Fernique, « AutoWIG : automatisation de l'encapsulation de bibliothèques 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>
- [28] 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>
- [29] « Reproducible Builds ». Consulté le: 4 septembre 2025. [En ligne]. Disponible sur: <https://reproducible-builds.org/>
- [30] 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>
- [31] « 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>
- [32] « 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>
- [33] « 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/>
- [34] 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>
- [35] Consulté le: 19 mai 2025. [En ligne]. Disponible sur: <https://nix.dev/manual/nix/2.17/language/>
- [36] Consulté le: 3 septembre 2025. [En ligne]. Disponible sur: <https://search.nixos.org/packages>
- [37] NixOS Wiki Authors, « Nixpkgs/Contributing ». Consulté le: 3 septembre 2025. [En ligne]. Disponible sur: <https://wiki.nixos.org/wiki/Nixpkgs/Contributing>
- [38] « Cachix — Nix binary cache hosting ». Consulté le: 3 septembre 2025. [En ligne]. Disponible sur: <https://www.cachix.org/>
- [39] « Nix (package manager) — Sandboxing ». Consulté le: 3 septembre 2025. [En ligne]. Disponible sur: [https://wiki.nixos.org/wiki/Nix\\_\(package\\_manager\)#Internals](https://wiki.nixos.org/wiki/Nix_(package_manager)#Internals)
- [40] « GitHub-hosted runners », *Github*. Consulté le: 4 septembre 2025. [En ligne]. Disponible sur: <https://docs.github.com/en/actions/concepts/runners/github-hosted-runners>
- [41] Consulté le: 6 juin 2025. [En ligne]. Disponible sur: <https://www.raspberrypi.com/>

- [42] 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>
- [43] « Eclipse Cyclone DDS - Home ». Consulté le: 17 octobre 2025. [En ligne]. Disponible sur: <https://cyclonedds.io/>
- [44] 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/>
- [45] 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>
- [46] 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](https://github.com/unitreerobotics/unitree_sdk2/blob/main/%7B2025-10-17%7D/include/unitree/idl/hg/LowCmd_.hpp#L33-L63)
- [47] 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>
- [48] 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](https://github.com/unitreerobotics/unitree_sdk2/tree/main/%7B2025-10-17%7D)
- [49] « 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](https://cmake.org/cmake/help/latest/prop_tgt/IMPORTED_LOCATION.html)
- [50] « Ghidra - Powerful Open-Source Reverse Engineering Tool ». Consulté le: 17 octobre 2025. [En ligne]. Disponible sur: <https://ghidralite.com/>
- [51] Unitree Robotics, *unitree\_mujoco*. (1 novembre 2021). GitHub. Consulté le: 17 octobre 2025. [En ligne]. Disponible sur: [https://github.com/unitreerobotics/unitree\\_mujoco](https://github.com/unitreerobotics/unitree_mujoco)
- [52] « 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](https://cyclonedds.io/docs/cyclonedds/latest/getting_started/helloworld/helloworld_source_code.html)
- [53] Unitree Robotics, *unitree\_ros2*. (20 octobre 2023). GitHub. Consulté le: 22 octobre 2025. [En ligne]. Disponible sur: [https://github.com/unitreerobotics/unitree\\_ros2](https://github.com/unitreerobotics/unitree_ros2)
- [54] « SDFFormat Specification, world.plugin element ». Consulté le: 22 octobre 2025. [En ligne]. Disponible sur: [http://sdformat.org/spec?ver=1.12&elem=world#world\\_plugin](http://sdformat.org/spec?ver=1.12&elem=world#world_plugin)
- [55] « Gazebo Sim: Finding resources ». Consulté le: 22 octobre 2025. [En ligne]. Disponible sur: [https://gazebo.org/api/sim/8/resources.html#:~:text=All%20paths%20on%20the%20GZ\\_SIM\\_SYSTEM\\_PLUGIN\\_PATH%20environment%20variable](https://gazebo.org/api/sim/8/resources.html#:~:text=All%20paths%20on%20the%20GZ_SIM_SYSTEM_PLUGIN_PATH%20environment%20variable)
- [56] « SDFFormat 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](http://sdformat.org/spec?ver=1.12&elem=world#plugin_filename)

## A Preuves

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

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

$$\forall s \in S, Q(s, 1) = Q(s, 2) \quad (25)$$

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

$$\forall s \in S, Q'(s, 1) := 2Q(s, 1) \quad (26)$$

$$\forall s \in S, Q'(s, 2) := \frac{1}{2}Q(s, 2) \quad (27)$$

$$\forall s \in S, \forall a \in A - \{1, 2\}, Q'(s, a) := Q(s, a) \quad (28)$$

On a

$$D_{\text{KL}}(Q \parallel Q') = \sum_{(s,a) \in S \times A} Q(s, a) \log \frac{Q(s, a)}{Q'(s, a)}$$

On découpe la somme selon les valeurs de  $A$ :

$$\begin{aligned} &= \sum_{s \in S} \sum_{a \in A - \{1, 2\}} \left[ Q(s, a) \log \frac{Q(s, a)}{Q'(s, a)} \right] + Q(s, 1) \log \frac{Q(s, 1)}{Q'(s, 1)} + Q(s, 2) \log \frac{Q(s, 2)}{Q'(s, 2)} \\ &= \sum_{s \in S} \underbrace{\sum_{a \in A - \{1, 2\}} Q(s, a) \log \frac{Q(s, a)}{Q(s, a)}}_{\text{d'après (28)}} + Q(s, 1) \log \underbrace{\frac{Q(s, 1)}{2Q(s, 1)}}_{\text{d'après (26)}} + Q(s, 2) \log \underbrace{\frac{Q(s, 2)}{\frac{1}{2}Q(s, 2)}}_{\text{d'après (27)}} \\ &= \sum_{s \in S} Q(s, 1) \left[ \log Q(s, 1) - \log Q(s, 1) - \log 2 \right] + \quad (29) \\ &\quad Q(s, 2) \left[ \log Q(s, 2) - \log Q(s, 2) - \log \frac{1}{2} \right] \\ &= \sum_{s \in S} -Q(s, 1) \log 2 + Q(s, 2) \log 2 \\ &= \sum_{s \in S} \log 2 \underbrace{(Q(s, 2) - Q(s, 1))}_{\text{d'après (25)}} \\ &= \sum_{s \in S} 0 = 0 \end{aligned}$$

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

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

On a

$$\begin{aligned} \exp\left(\sum_{t=0}^{\infty} \gamma^t r(C_t)\right) &= \sum_{(c_t)_{t \in \mathbb{N}} \in \mathcal{S}} \left(\sum_{t=0}^{\infty} \gamma^t r(c_t)\right) \mathbb{P}\left(\sum_{t=0}^{\infty} \gamma^t r(C_t) = \sum_{t=0}^{\infty} \gamma^t r(c_t)\right) \\ &= \sum_{(c_t)_{t \in \mathbb{N}} \in \mathcal{S}} \left(\sum_{t=0}^{\infty} \gamma^t r(c_t)\right) \mathbb{P}(C = (c_t)_{t \in \mathbb{N}}) \end{aligned} \quad (30)$$

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

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  $(c_t)_{t \in \mathbb{N}} \in \mathcal{S}$  :

$$\begin{aligned} P(C = (c_t)_{t \in \mathbb{N}}) &= \\ \mathbb{P}(S_0 = s_0) \mathbb{P}(A_0 = a_0 \mid S_0 = s_0) \prod_{t=1}^{\infty} \mathbb{P}(S_t = s_t \mid C_{t-1} = c_{t-1}) \mathbb{P}(A_t = a_t \mid S_t = s_t) \end{aligned} \quad (31)$$

On a

$$\begin{aligned} \mathbb{P}(S_0 = s_0) &= \rho_0(s_0) \\ \forall t \in \mathbb{N}, \quad \mathbb{P}(A_t = a_t \mid S_t = s_t) &= Q_p(s_t, a_t) \\ \forall t \in \mathbb{N}^*, \quad \mathbb{P}(S_t = s_t \mid C_{t-1} = c_{t-1}) &= \mathbb{P}(M(C_{t-1}) = M(c_{t-1}) \mid C_{t-1} = c_{t-1}) \\ &= \mathbb{P}(C_{t-1} = c_{t-1} \mid C_{t-1} = c_{t-1}) = 1 \end{aligned} \quad (32)$$

Donc on a

$$\begin{aligned} P(C = (c_t)_{t \in \mathbb{N}}) &= \rho_0(s_0) Q_p(s_0, a_0) \prod_{t=1}^{\infty} Q_p(s_t, a_t) \\ &= \rho_0(s_0) \prod_{t=0}^{\infty} Q_p(s_t, a_t) \end{aligned} \quad (33)$$

Et ainsi

$$\begin{aligned} \exp\left(\sum_{t=0}^{\infty} \gamma^t r(C_t)\right) &= \sum_{(c_t)_{t \in \mathbb{N}} \in \mathcal{S}} \left(\sum_{t=0}^{\infty} \gamma^t r(c_t)\right) \mathbb{P}(C = (c_t)_{t \in \mathbb{N}}) \\ &= \sum_{(c_t)_{t \in \mathbb{N}} \in \mathcal{S}} \left(\sum_{t=0}^{\infty} \gamma^t r(c_t)\right) \rho_0(s_0) \prod_{t=0}^{\infty} Q_p(s_t, a_t) \\ &= \eta(p, r) \quad \blacksquare \end{aligned} \quad (34)$$

**A.3 Simplification de l'expression de  $L(s, a, \mathcal{P}, \mathcal{P}', R)$  dans PPO-Clip**

Soit  $(s, a) \in S \times A$ , et  $\mathcal{P}'$  une politique. Posons  $\alpha := A_{\mathcal{P}', R}(s, a)$ ,  $q/q' := Q_{\mathcal{P}}(s, a)/Q_{\mathcal{P}'}(s, a)$ .

**Cas  $\alpha > 0$** 

$$L(s, a, \mathcal{P}, \mathcal{P}', R)$$

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

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

$$\text{car } \alpha > 0$$

**Cas  $\alpha < 0$** 

$$L(s, a, \mathcal{P}, \mathcal{P}', R)$$

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

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

$$\text{car } \alpha < 0$$

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

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

$$= \min\left(\frac{q}{q'}, \quad \frac{q}{q'}\right)\alpha$$

$$= \min\left(\frac{q}{q'}, 1 + \varepsilon\right)\alpha$$

$$\text{car } 1 + \varepsilon > \frac{q}{q'}$$

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

$$= \max\left(\frac{q}{q'}, \quad \frac{q}{q'}\right)\alpha$$

$$= \max\left(\frac{q}{q'}, 1 - \varepsilon\right)\alpha$$

$$\text{car } 1 - \varepsilon < \frac{q}{q'}$$

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

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

$$= \min\left(\frac{q}{q'}, \quad 1 + \varepsilon\right)\alpha$$

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

$$= \max\left(\frac{q}{q'}, \quad 1 + \varepsilon\right)\alpha$$

$$= \max\left(\frac{q}{q'}, \quad 1 - \varepsilon\right)\alpha$$

$$\text{car } 1 - \varepsilon < 1 + \varepsilon < \frac{q}{q'}$$

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

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

$$= \min\left(\frac{q}{q'}, \quad 1 - \varepsilon\right)\alpha$$

$$= \min\left(\frac{q}{q'}, \quad 1 + \varepsilon\right)\alpha$$

$$\text{car } 1 + \varepsilon > 1 - \varepsilon > \frac{q}{q'}$$

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

$$= \max\left(\frac{q}{q'}, \quad 1 - \varepsilon\right)\alpha$$