Navigation autonome de bout en bout par technique d'apprentissage par renforcement

Gwendal PRISER

Gwendal.PRISER@ensta-bretagne.org

Rapport de projet de fin d'étude
Dîplome d'ingénieur généraliste de l'ENSTA Bretagne
Spécialité Robotique Mobile

3 Mars 2021 - 3 Septembre 2021

Tuteur école :
Luc Jaulin
luc.jaulin@ensta-bretagne.fr

Tuteurs CEA :
Karim Ben Chehida
karim.benchehida@cea.fr
Dinh Khanh Ho
dinhkhanh.ho@cea.fr

TITRE/*TITLE* : NAVIGATION AUTONOME DE BOUT EN BOUT PAR TECHNIQUE D'APPRENTISSAGE PAR RENFORCEMENT	DRT/LIST/DSCIN/LIAE
AUTEUR/*AUTHOR* : GWENDAL PRISER	
UNITE/*UNIT* :DRT/LIST/DSCIN/LIAE	Page 2
PROJET/*PROJECT* :	

NOMBRE DE PAGES/*NUMBER OF PAGES* : 48

RESUME/*SUMMARY* :

La navigation est une des fonctionnalités les plus importantes des systèmes mobiles autonomes qui permet au véhicule de se déplacer d'un point à l'autre. Dans un système de navigation, il y a souvent deux niveaux de planificateur : un planificateur global pour donner des directives/des objectifs de navigation (par exemple, une trajectoire entre deux points planifiés basée sur une carte globale de l'environnement) et un planificateur local pour suivre ces directives/objectifs donnés par le planificateur global, calculer un cap et une vitesse en sortie, en détectant et évitant des obstacles. Récemment, il y a des approches qui appliquent la technique d'apprentissage de bout en bout pour améliorer le planificateur local de navigation autonome. L'objectif de ce rapport est d'explorer l'état de l'art et de proposer un mécanisme de navigation des systèmes mobiles autonomes plus intelligent et plus autonome basé sur la technique d'apprentissage par renforcement profond.

MOTS CLES/*KEYWORDS* : robotique mobile – navigation de bout-en-bout – apprentissage par renforcement – perception – prise de décision

	Rédacteur/*Editor*	Vérificateur/*Checked by*	Approbateur/*Approved by*
NOM/*NAME*	Gwendal PRISER	Karim Ben Chehida	Maroun OUAIL
DATE/*DATE*	30/08/2021	30/08/21	30/08/2021
VISA/*VISA*			

Ce document et les informations qu'il contient sont la propriété exclusive du CEA.
This document and the information it contains are the exclusive property of CEA.
Ils ne peuvent pas être communiqués sans une autorisation préalable du LIST./
They cannot be communicated without prior authorization from LIST.

Table des matières

1 Introduction **4**
1.1 Aperçu historique et motivations . . . 4
1.2 Problématique . . . 4
1.3 Organisation du rapport . . . 6

2 Présentation de l'institut de recherche **7**
2.1 CEA . . . 7
2.2 CEA List . . . 7
2.3 LIAE . . . 8

3 Prérequis théoriques et état de l'art **9**
3.1 Navigation autonome . . . 9
3.2 Introduction à l'apprentissage par renforcement . . . 10
3.3 Processus de décision markovien . . . 10
3.4 Algorithmes de l'apprentissage par renforcement . . . 11
3.4.1 Méthodes basées sur la fonction de valeur . . . 11
3.4.2 Méthodes basées sur l'optimisation et recherche directe de la politique . . . 13
3.4.3 Acteur - Critique . . . 15
3.5 Robotique mobile et apprentissage par imitation . . . 16
3.6 Robotique mobile et apprentissage par renforcement . . . 17
3.6.1 Apprentissage par renforcement avec données LIDAR pour de la navigation . 17
3.6.2 Apprentissage par renforcement avec données visuelles pour de la navigation 18
3.7 Multi-modalité, perception de l'environnement et transfert . . . 19
3.7.1 Multi-modalité . . . 20
3.7.2 Perception de l'environnement et transfert . . . 20
3.8 Synthèse de l'état de l'art . . . 21
3.8.1 Espace d'état . . . 21
3.8.2 Fonction de récompense . . . 21
3.8.3 Espace d'actions . . . 21
3.8.4 Algorithme . . . 22
3.8.5 Simulateur . . . 22

4 Algorithme et méthode d'entraînement utilisés **23**
4.1 Proximal Policy Optimisation(PPO) . . . 23
4.2 Architecture de l'algorithme . . . 24
4.3 Méthode d'entraînement . . . 26
4.3.1 Simulateurs . . . 27
4.3.2 Fonction de récompense . . . 30
4.3.3 Implémentation de l'algorithme . . . 31
4.4 Résultats de l'entraînement dans Habitat . . . 31
4.4.1 Métriques d'évaluation . . . 31
4.4.2 Serveur utilisé pour l'entraînement . . . 32
4.4.3 Résultats et comparaisons des fonctions de la récompense . . . 32

4.4.4 Inférence et intérêt de la multi-modalité : RGB + D . . . 38
4.4.5 Inférence et intérêt de l'optimisation du *backbone* . . . 38
4.4.6 Hyperparamètres de l'algorithme PPO . . . 39

5 Expérimentations et évaluations **40**
5.1 Transfert dans Gazebo . . . 40
5.1.1 Description matérielle et modélisation de l'agent . . . 40
5.1.2 Architecture logicielle . . . 40
5.1.3 Comportement de l'agent . . . 41
5.2 Transfert dans le monde réel . . . 42
5.2.1 Architecture du monde . . . 42
5.2.2 Méthode utilisée . . . 43
5.2.3 Scénarios mis-en-place et résultats de navigation . . . 44

6 Conclusion et perspectives **45**
6.1 Conclusion . . . 45
6.2 Perspectives . . . 45

1 Introduction

1.1 Aperçu historique et motivations

Réseaux de neurones Les réseaux de neurones existent depuis les années 1950, mais ils ont gagné en popularité à partir des années 2010 avec le développement du Deep-Learning. Cela a été rendu possible avec d'une part l'augmentation de la puissance de calcul des ordinateurs, et en particulier des GPU permettant de paralléliser le nombre d'opérations. D'autre part, des datasets annotées comme ImageNet ont permis au Deep-Learning de surpasser les techniques classiques de vision par ordinateur. Le Deep-Learning a de nombreuses applications, il est très utilisé en vision par ordinateur ou en traitement du signal. Dans ce rapport, nous nous attarderons sur l'application du Deep-Learning à la prise de décision d'un robot. Cette application est encore marginale mais gagne en popularité depuis 2016 avec la victoire d'un algorithme de deep-learning face au multiple champion du monde de Go.

Robotique mobile Les robots évoluent rapidement, passant d'environnements industriels, physiquement fixés sur leur lieu de travail, à des machines de plus en plus complexes capables d'effectuer des tâches difficiles dans notre environnement quotidien (figure 1). Les robots industriels traditionnels utilisés dans les usines de fabrication où l'environnement est hautement contrôlé sont généralement plus ou moins stationnaires. En revanche, les robots mobiles sont des systèmes robotisés qui peuvent fonctionner dans des environnements sans contraintes et qui ont la capacité de se déplacer librement en utilisant leurs roues. Ils peuvent fonctionner de manière autonome dans un environnement partiellement inconnu et imprévisible sans avoir besoin de dispositifs de guidage physiques.

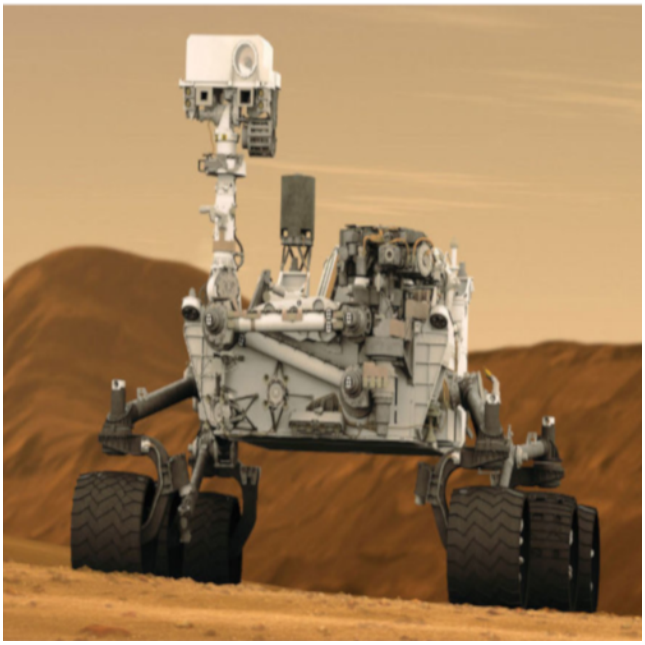
Dans ce rapport, nous nous concentrons sur ces robots mobiles à roues autonomes, c'est-à-dire les robots qui sont capables de prendre leurs propres décisions en fonction de la situation dans laquelle ils se trouvent sans utiliser une suite prédéfinie de commandes et mouvements.

Les progrès réalisés dans le domaine des robots mobiles autonomes en terme d'intégration du calcul, de batteries et de capteurs ont également apporté des solutions à des tâches complexes qui étaient auparavant considérées comme réalisables que par des humains. Ces domaines comprennent l'exploration planétaire ou sous-marine, l'exploration en milieu urbain, etc (figure 1). Ces robots autonomes peuvent même dépasser les capacités humaines en explorant, par exemple, plus longtemps et plus profondément les océans.

1.2 Problématique

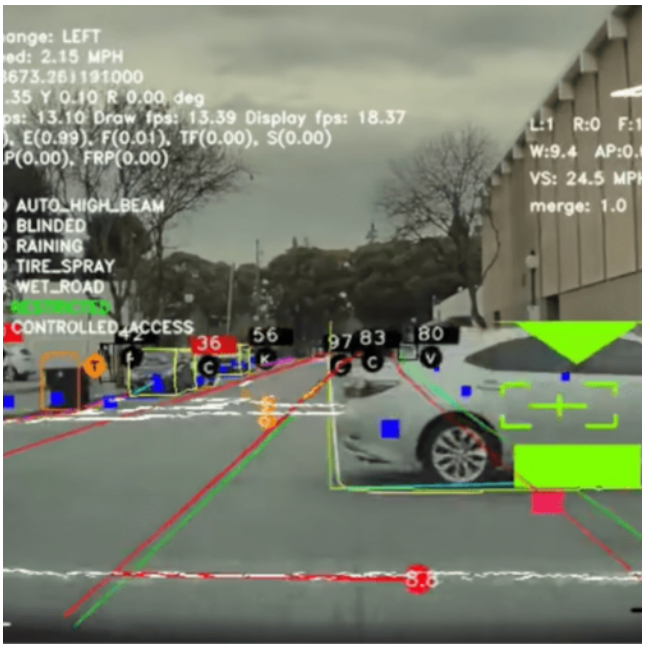
A la lumière de ces observations, nous souhaitons développer un algorithme utilisant des méthodes issues de l'apprentissage par renforcement et qui permet à un robot mobile à roues de naviguer en autonomie. Nous lui fixons comme contraintes de ne pas pouvoir utiliser une carte de son environnement, et d'évoluer en intérieur.

Par ailleurs, l'algorithme doit pouvoir être embarqué sur un robot réel et permettre au robot de naviguer en sécurité, i.e. il ne percute pas d'obstacles et peut adapter en temps réel sa navigation si des obstacles dynamiques apparaissent sur sa trajectoire. En effet, connaissant la carte de leur environnement les robots mobiles savent parfaitement naviguer mais ils ont des difficultés à adapter

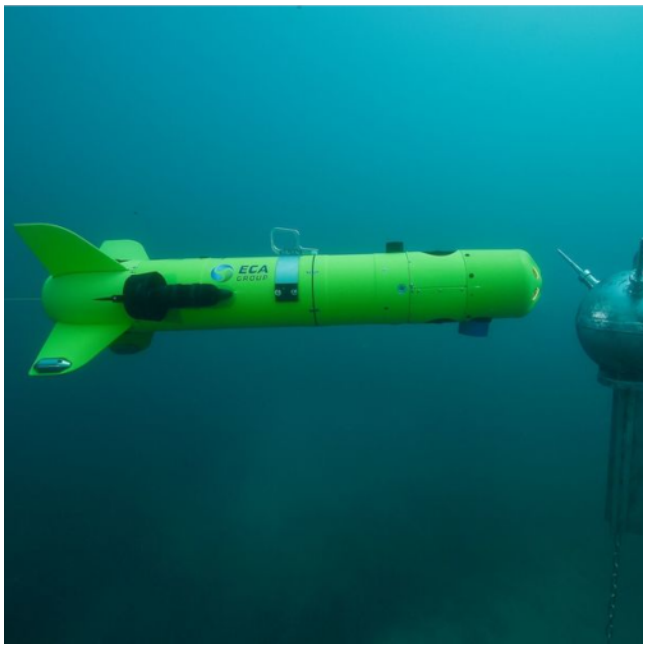


(a) Curiosity (NASA) en exploration sur Mars

(b) Robot mobile dans les centres de tri d'Amazon



(c) Segmentation et labélisation d'une image réelle pour de la conduite autonome



(d) AUV en exploration sous-marine pour neutraliser des mines

FIGURE 1: Exemples d'utilisation de la robotique mobile

leur navigation si des obstacles se placent sur leurs parcours. Embarquer l'algorithme, le contraint également sur les ressources de calcul qu'il peut utiliser.

1.3 Organisation du rapport

Nous présenterons d'abord le CEA et le laboratoire dans lequel s'est déroulé le stage. Nous dresserons ensuite un état de l'art centré sur la problématique ainsi que la présentation des prérequis théoriques pour comprendre les travaux. En troisième partie, nous aborderons la méthodologie et l'implémentation de l'algorithme. Enfin nous présenterons nos résultats d'expérience.

2 Présentation de l'institut de recherche

Le stage s'est déroulé du 3 mars 2021 au 3 septembre 2021 au laboratoire LIAE appartenant au CEA de Saclay. Il est placé dans l'antenne Nano-Innov à Palaiseau située à 6 km du site historique de Saclay. La majeure partie du travail a été effectuée en présentiel avec l'aide précieuse de mes tuteurs.

2.1 CEA

Le Commissariat à l'Energie Atomique et aux Energies Alternatives (CEA) est un organisme public français de recherche à caractère scientifique, technique et industriel. Ses principaux axes de recherche se focalisent dans les domaines de l'énergie, des technologies de l'information et de la communication, des sciences de la matière et des sciences de la vie et de la santé. Il possède neuf centres de recherche éparpillés dans toute la France dont un dans la grappe industrielle technologique Paris-Saclay et un dans le Polygone scientifique de Grenoble.

Le CEA fut fondé dès la fin de la Seconde guerre mondiale en octobre 1945 par le général de Gaulle, qui voulait mettre en place un organisme de recherche consacré à l'énergie atomique suite au largage des deux bombes atomiques américaines sur Hiroshima et Nagasaki. Le CEA se retrouve alors dirigé par Frédéric Joliot-Curie (physicien et chimiste français ayant obtenu avec son épouse Irène le prix Nobel de chimie en 1935) et par Raoul Dautry, ministre de la reconstruction et de l'urbanisme. L'unique but de cet organisme à l'époque était de poursuivre des recherches scientifiques et techniques avec pour objectif l'utilisation de l'énergie nucléaire dans les applications médicales, industrielles (production d'électricité en outre) et dans la défense nationale. Depuis les années 2000, le CEA diversifie ses activités dans les énergies renouvelables (énergie solaire, piles à combustibles et à hydrogène, biocarburants) et dans les nouvelles technologies (ordinateur quantique, intelligence artificielle, voitures autonomes, robotique).

2.2 CEA List

Le Laboratoire d'Intégration des Systèmes et des Technologies est l'un des trois instituts de recherche technologique du CEA. Basé à Saclay en Essonne, il se dédie aux systèmes numériques intelligents, sa mission étant de réaliser des développements technologiques d'excellence pour le compte de partenaires industriels. Les principaux projets de recherche se concentrent essentiellement dans quatre domaines : le manufacturing avancé, les systèmes embarqués, la data intelligence et la maîtrise des rayonnements pour la santé. Le List s'inscrit également dans un environnement national particulièrement dynamique, avec des partenariats stratégiques au niveau académique (Paris-Saclay étant la 13ème meilleure université au monde en 2021 [1]), collaboratif et industriel. De plus, l'institut est aussi engagé dans plus de 200 projets collaboratifs européens, collabore avec des laboratoires académiques étrangers de premier plan et mène des actions de recherche au niveau international.

1. https://www.shanghairanking.com/rankings/arwu/2021

2.3 LIAE

L'institut CEA-LIST est constitué de 5 départements dont le DSCIN "Département des Systèmes et Circuits Intégrés Numériques" qui est un département spécial du LIST puisqu'à cheval entre SACLAY et Grenoble pour optimiser les interactions entre les instituts LIST et LETI. Ce département comprend 6 laboratoires :

- A Saclay
  - LECA : Environnement de Conception et Architecture
  - *LIAE : Intelligence Artificielle Embarquée*
  - LCYL : Confiance de sYstèmes de calcuL
- A Grenoble
  - LSTA : Systèmes-sur-puce et Techonolgies Avancées
  - LFIM : Fonctions Inovantes pour circuits Mixtes
  - LIIM : Intelligence Intégrée Multi-capteurs

Le laboratoire LIAE est le plus grand laboratoire du DSCIN, il est constitué de 26 ingénieurs chercheurs, 5 doctorants et 4 stagiaires. Ce laboratoire a comme principaux axes de recherche :

- L'intégration de l'IA dans des systèmes embarqués (outils de modélisation et conception matérielle d'accélérateurs d'IA)
- Imageurs intelligents et traitement au plus proche du capteur (analyse et compréhension avancée de la scène pour un traitement dans le capteur)
- Perception efficace de confiance (co-conception logicielle matérielle de systèmes de perception par fusion multimodale sous contraintes)
- Systèmes mobiles autonomes (Optimisation de l'énergie et de la qualité de services des missions, navigation autonome)

Mes activités ont principalement été effectuées dans le cadre de l'axe de recherche "Systèmes mobiles Autonomes" du LIAE en contribuant à développer et valider des concepts liés à la navigation robotique autonome.

3 Prérequis théoriques et état de l'art

Nous dresserons un état de l'art sur les méthodes de navigation pour les robots mobiles à roues et nous présenterons les prérequis théoriques nécessaires à la compréhension des travaux.

3.1 Navigation autonome

De manière classique dans le domaine de la conduite autonome, nous pouvons distinguer deux approches.

Approche modulaire L'approche classique est l'approche modulaire. Elle est constituée d'une suite de modules effectuant des tâches intermédiaires pour la navigation, comme par exemple cette séquence (figure 2) : un module de perception, un module de planification de trajectoires et enfin un module de contrôle chargé d'envoyé les commandes aux actionneurs pour que le véhicule approche au mieux la trajectoire planifiée. De cette manière, nous pouvons suivre individuellement chaque étape de la prise de décision, le comportement du robot est plus interprétable. Toutefois l'optimisation est difficile car les modules sont interdépendants et nous ne pouvons optimiser un module qu'indépendamment des autres.

Approche de bout-en-bout La deuxième approche qui devient de plus en plus populaire est la conduite autonome de bout-en-bout (figure 2). Dans cette approche le système autonome prend en entrée des données de capteurs comme par exemple une image RGB, une carte de profondeur, une position GPS, etc., et ressort directement les instructions pour le contrôleur du véhicule : direction et accélération. L'avantage de cette méthode est d'éliminer les étapes intermédiaires, nous supprimons ainsi le biais humain inhérent à la construction d'un système autonome modulaire. De plus, avec une approche de bout-en-bout, il est plus facilement possible de s'affranchir du planificateur global lié à la carte de l'environnement. La navigation peut être dynamique mais ces approches souffrent souvent d'un manque de généralisation et sont plus difficilement interprétables.

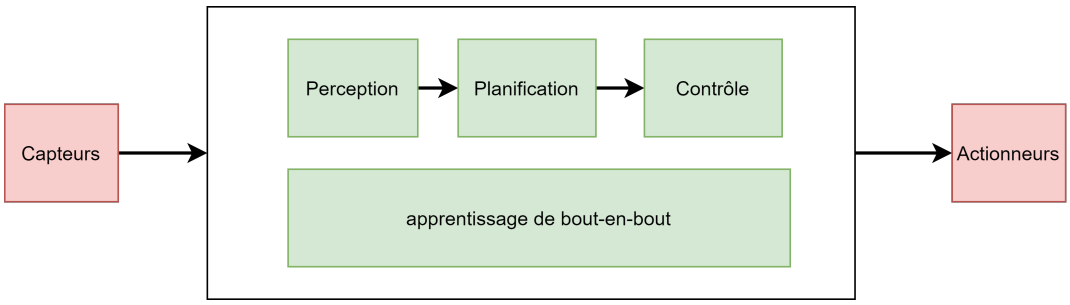


FIGURE 2: Approche modulaire et approche de bout-en-bout

Dans notre état de l'art, nous nous concentrerons sur l'approche de bout-en-bout, et plus particulèrement sur les algorithmes d'apprentissage par renforcement utilisant des réseaux neurones.

3.2 Introduction à l'apprentissage par renforcement

Dans cette section nous introduisons les concepts de l'apprentissage par renforcement, nécessaires pour comprendre l'état de l'art ainsi que nos travaux.

3.3 Processus de décision markovien

Formellement l'apprentissage par renforcement peut être décrit comme un processus de décision markovien (MDP), qui est un tuple constitué des éléments suivants (cf. figure 3) :

- Un ensemble d'états $\mathcal{S}$, plus une distribution d'ensemble de départ $p(\mathbf{s}_0)$.
- Un ensemble d'actions $\mathcal{A}$.
- Une matrice de transitions : $\mathcal{T}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ qui associe un couple action/état à un instant t avec une distribution d'états à un instant $t+1$.
- Une fonction immédiate de la récompense. $\mathcal{R}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$.
- Un facteur de réduction $\gamma \in [0, 1]$, qui met en valeur les récompenses immédiates et appauvrit les récompenses tardives

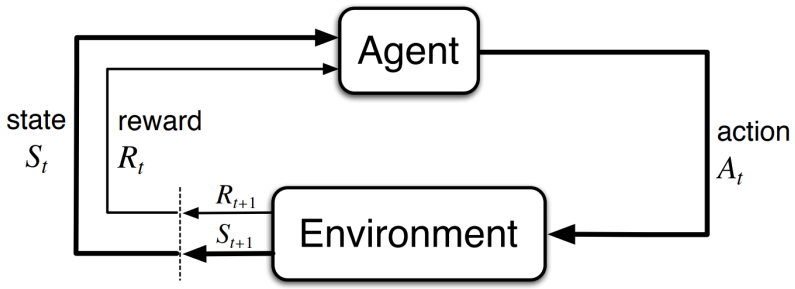


FIGURE 3: L'interaction de l'agent avec son environnement au sein d'un processus de décision markovien d'après Sutton and Barto [19]

En général, la politique π est la fonction qui associe des états à une distribution de probabilité sur des actions : $\pi : \mathcal{S} \rightarrow p(\mathcal{A} = \mathbf{a}|\mathcal{S})$.

Si le processus de décision markovien est *épisodique*, i.e., les états sont remis à zero après chaque épisode de longueur T, cette séquence d'états, actions, récompenses sur un épisode est appelée une *trajectoire* ou *rollout* associée à la politique. Chaque trajectoire d'une politique accumule les récompenses provenant de l'environnement, résultant en un retour $R = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$. Les récompenses obtenues à court-terme sont bien plus valorisées que les récompenses obtenues tardivement (γ^t) Le but de l'appentissage par renforcement est de trouver une politique optimale, π^*, réussissant à maximiser l'espérance du retour sur l'ensemble des états :

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}[R|\pi]. \tag{1}$$

Il est aussi possible de considérer un processus de décision markovien non-épisodique, où $T = \infty$. Dans cette situation, $\gamma < 1$ empêche une somme infinie de récompenses d'être accumulée.

Un concept clé propre à l'apprentissage par renforcement est la propriété de Markov : seulement l'état courant peut affecter l'état suivant, ou encore il suffit de connaître l'état présent pour connaître l'état suivant. Cela veut dire que chaque décision prise en l'état $\mathbf{s}_t$ peut être entièrement basée sachant $\mathbf{s}_{t-1}$, à la place de $\{\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_{t-1}\}$. Bien que cette hypothèse soit effectuée dans une majorité des algorithmes d'apprentissage par renforcement, souvent cela n'est pas réaliste car cela requiert que l'environnement soit *entièrement observable*. Une généralisation des processus de décision markoviens sont les processus de décision markovien partiellement observables (POMDPs), dans lequel l'agent reçoit une observation $\mathbf{o}_t \in \Omega$, où la distribution des observations $p(\mathbf{o}_{t+1}|\mathbf{s}_{t+1}, \mathbf{a}_t)$ est dépendante de l'état courant et de l'action précédente.

Les algorithmes POMDP maintiennent une *croyance* sur les états présents étant donné une croyance sur les états précédents, l'action effectuée et l'observation courante. Dans ce cas là, une approche courante en apprentissage profond est d'utiliser les réseaux de neurones récurrents (RNN)

A la différence des réseaux *feed-forward*, les réseaux de neurones récurrents sont dynamiques. Cela aide quand l'état ne peut être que partiellement estimé.

3.4 Algorithmes de l'apprentissage par renforcement

Jusqu'à présent, nous avons présenté le formalisme clé utilisé en apprentissage par renforcement et le processus de décision markovien. Dans ce qui suit, nous allons distinguer différentes classes d'algorithmes d'apprentissage par renforcement. Il existe deux approches principales pour résoudre les problèmes d'apprentissage par renforcement : les méthodes basées sur les fonctions de valeur et les méthodes basées sur la recherche de politique. Il existe également une approche hybride, l'approche acteur-critique, qui utilise à la fois les fonctions de valeur et la recherche de politique. Ces méthodes d'acteur-critique constituent l'état de l'art des algorithmes d'apprentissage par renforcement. Nous allons maintenant expliquer ces approches et d'autres concepts utiles pour la résolution des problèmes d'apprentissage par renforcement.

3.4.1 Méthodes basées sur la fonction de valeur

Les méthodes basées sur la fonction de valeur consistent à estimer la valeur qu'on a, à être dans un certain état. La *state-value function* $V^\pi(\mathbf{s})$ correspond à l'espérance du retour en partant d'un état $\mathbf{s}$ et en suivant la politique π :

$$V^\pi(\mathbf{s}) = \mathbb{E}[R|\mathbf{s}, \pi] \tag{2}$$

La politique optimale, π^*, a une *state-value function* (SVF) qui lui correspond $V^*(\mathbf{s})$, et vice-versa, la SVF optimale peut être décrite comme

$$V^*(\mathbf{s}) = \max_\pi V^\pi(\mathbf{s}) \quad \forall \mathbf{s} \in \mathcal{S}. \tag{3}$$

Si nous avions $V^*(\mathbf{s})$ disponible, la politique optimale pourrait être trouvée en choisissant parmi les actions disponibles dans l'état $\mathbf{s}_t$ et en prenant l'action $\mathbf{a}$ qui maximise

$\mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{T}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a})}[V^*(\mathbf{s}_{t+1})]$.

Dans le cadre de l'apprentissage par renforcement, les matrices de transitions $\mathcal{T}$ ne sont pas disponibles. Donc nous construisons une autre fonction, la *state-action-value* (SAVF) ou *fonction de la qualité* $Q^\pi(\mathbf{s}, \mathbf{a})$, qui ressemble à V^π, sauf que l'on donne une action initiale $\mathbf{a}$, et une politique π dont dépendra les actions choisies dans les états suivants :

$$Q^\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}[R|\mathbf{s}, \mathbf{a}, \pi]. \tag{4}$$

La meilleure politique, étant donné $Q^\pi(\mathbf{s}, \mathbf{a})$, peut être trouvée en choisissant $\mathbf{a}$ de manière avide à chaque état : $\text{argmax}_\mathbf{a}\, Q^\pi(\mathbf{s}, \mathbf{a})$. Avec la politique , nous pouvons aussi définir $V^\pi(\mathbf{s})$ en maximisant $Q^\pi(\mathbf{s}, \mathbf{a})$: $V^\pi(\mathbf{s}) = \max_\mathbf{a} Q^\pi(\mathbf{s}, \mathbf{a})$.

Programmation dynamique : Pour apprendre la *state-action-value* Q^π, Nous exploitons la propriété de Markov et définissons la *state-action-value* comme une équation de Bellman qui a la forme récursive suivante

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{\mathbf{s}_{t+1}}[r_{t+1} + \gamma Q^\pi(\mathbf{s}_{t+1}, \pi(\mathbf{s}_{t+1}))]. \tag{5}$$

Cela veut dire que Q^π peut être améliorée par *bootstrapping*, i.e., nous pouvons utiliser la valeur de notre estimation de Q^π pour améliorer notre estimation. Il s'agit du principe du Q-learning et de l'algorithme SARSA (état-action-reward-état-action) :

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) \leftarrow Q^\pi(\mathbf{s}_t, \mathbf{a}_t) + \alpha\delta, \tag{6}$$

où α est le *learning rate* et $\delta = Y - Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$ l'erreur de différence temporelle ; Y est la cible comme dans un problème standard de régression. SARSA, est un algorithme d'apprentissage *on-policy*, car les transitions sont estimées en prenant la politique actuelle et utilise :
$Y = r_t + \gamma Q^\pi(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})$.
Q-learning est *off-policy*, car les transitions n'utilisent pas forcément la politique actuelle. A la place , Q-learning utilise $Y = r_t + \gamma \max_\mathbf{a} Q^\pi(\mathbf{s}_{t+1}, \mathbf{a})$, qui fait une approximation directe de Q^*.

Dans des environnements complexes, Q-learning en programmation dynamique n'est pas envisageable car le nombre d'états à estimer est trop important. Nous pouvons alors collecter des trajectoires, et stocker les transitions dans un buffer. Nous effectuons alors seulement des différences temporelles (figure 4). Ce principe est utilisé par Mnih et al. [9] avec l'algorithme de Deep-Q-learning.

Itération de politique Pour trouver Q^* à partir d'un Q^π arbitraire, nous utilisons l'itération de politique généralisée, où l'itération de politique consiste en l'évaluation de la politique et l'amélioration de la politique. L'évaluation de la politique améliore l'estimation de la fonction de valeur, ce qui peut être réalisé en minimisant les erreurs de Bellman (δ) des trajectoires subies en suivant la politique. Au fur et à mesure que l'estimation s'améliore, la politique peut naturellement être améliorée en choisissant des actions avec avidité $\text{argmax}_\mathbf{a}\, Q^\pi(\mathbf{s}, \mathbf{a})$ en fonction de la fonction de valeur mise à jour (ou de manière ϵ-greedy en choisissant avec avidité la plupart du temps et ϵ

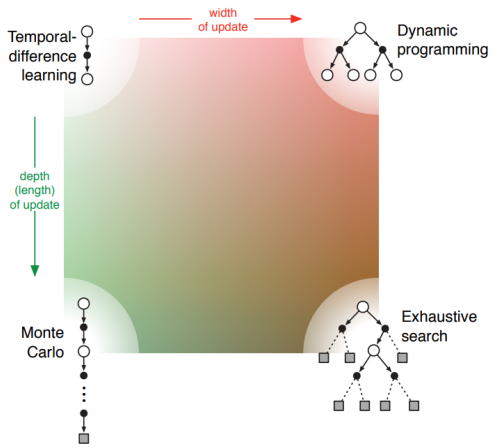


FIGURE 4: Programmation dynamique, TD-Learning et Monte-Carlo d'après Sutton and Barto [19]

fois sur un, l'action est choisie aléatoirement). Cela forme un processus de rétroactions positives (figure 5).

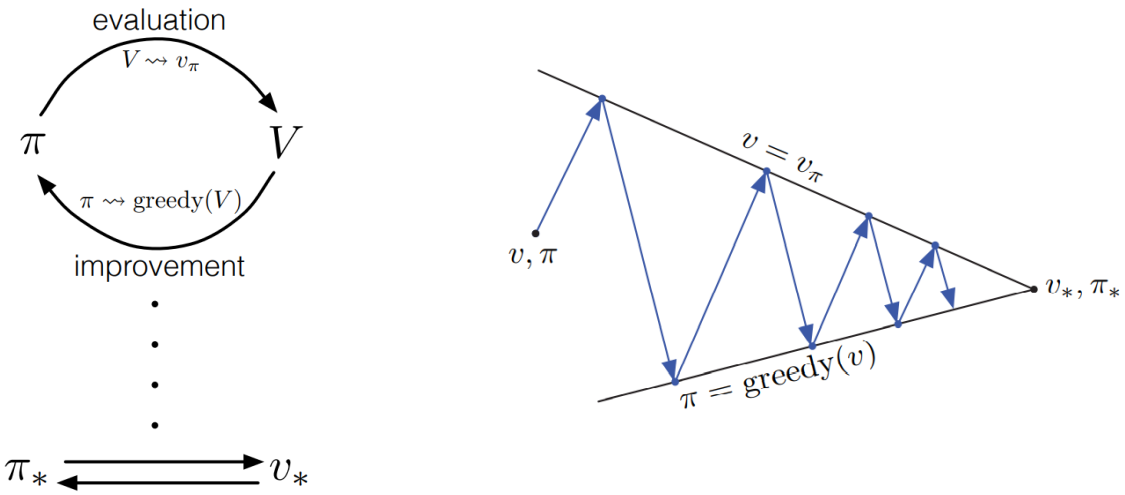


FIGURE 5: Principe de l'itération de politique Sutton and Barto [19]

3.4.2 Méthodes basées sur l'optimisation et recherche directe de la politique

A la différence des algorithmes basés sur l'optimisation de la fonction de valeur, comme le Q - Learning qu'on a pu voir avec la programmation dynamique ou dans sa version neuronale (Mnih et al. [9]), il existe des algorithmes qui visent à optimiser directement la politique. Nous allons voir le principe de cette classe d'algorithmes.

Algorithmes basés sur l'optimisation du gradient de la politique Considérons la trajectoire τ

$$\tau = (s_0, a_0, s_1, a_1, ...).$$

Nous voulons maximiser l'espérance sur le retour :

$$J(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) = \underset{\tau\sim\pi}{E}[R(\tau)]$$

Nous souhaitons optimiser la politique paramétrée par θ par une ascension de gradient :

$$\theta_{k+1} = \theta_k + \alpha \, \nabla_\theta J(\pi_\theta)|_{\theta_k}$$

On nomme l'expression suivante le gradient de la politique :

$$\nabla_\theta J(\pi_\theta)$$

Les algorithmes visant à optimiser la politique de cette manière sont nommés : *policy gradient algorithms*

Pour utiliser ces algorithmes nous avons besoin de l'expression du gradient de la politique. Quelques étapes de calculs sont nécessaires pour obtenir une expression du gradient de la politique estimable.

Probabilité de la trajectoire Nous pouvons définir la probabilité d'obtenir la trajectoire

$$\tau = (s_0, a_0, ..., s_{T+1})$$

ainsi :

$$P(\tau|\theta) = \rho_0(s_0) \prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

En utilisant la définition de la dérivée de la fonction logarithme nous effectuons le changement cosmétique suivant :

$$\nabla_\theta P(\tau|\theta) = P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)$$

Calculons le logarithme de la probabilité de la trajectoire :

$$\log P(\tau|\theta) = \log \rho_0(s_0) + \sum_{t=0}^{T} \Big(\log P(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t) \Big)$$

On rappelle que les termes suivants n'ont pas de dépendance en θ :
$\rho_0(s_0), \quad P(s_{t+1}|s_t, a_t), \quad R(\tau)$ donc leur dérivée est nulle.

$$\nabla_\theta \log P(\tau|\theta) = \cancel{\nabla_\theta \log \rho_0(s_0)} + \sum_{t=0}^{T} \Big(\cancel{\nabla_\theta \log P(s_{t+1}|s_t, a_t)} + \nabla_\theta \log \pi_\theta(a_t|s_t) \Big) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Ces étapes intermédiaires nous permettent de calculer le gradient de la politique $\nabla_\theta J(\pi_\theta)$:

$$
\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \underset{\tau\sim\pi_\theta}{E} R(\tau) \\
&= \nabla_\theta \int_\tau P(\tau|\theta)R(\tau) && \text{Développement de l'espérance} \\
&= \int_\tau \nabla_\theta P(\tau|\theta)R(\tau) && \text{Par linéarité du gradient} \\
&= \int_\tau P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)R(\tau) && \text{Astuce du passage au logarithme de la dérivée} \\
&= \underset{\tau\sim\pi_\theta}{E} \nabla_\theta \log P(\tau|\theta)R(\tau) && \text{Formule de l'espérance} \\
\nabla_\theta J(\pi_\theta) &= \underset{\tau\sim\pi_\theta}{E} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau)
\end{aligned}
$$

Maintenant le calcul du gradient de la politique ne dépend plus de $P(\tau|\theta)$ qui est inconnue car nous ne connaissons pas le modèle. Nous pouvons donc estimer le gradient de la politique sur un échantillon de trajectoires $\mathcal{D} = \{\tau_i\}_{i=1,\dots,N}$ et l'estimation de la trajectoire est la suivante :

$$\hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau\in\mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau)$$

avec $|\mathcal{D}|$ le nombre de trajectoires (ici N)

L'algorithme REINFORCE est l'algorithme le plus simple basé sur l'optimisation de la politique (c.f. Sutton and Barto [19]).

Algorithme 1 : REINFORCE d'après Sutton and Barto [19]

1: Entrées : Initialiser θ_0 arbitrairement
2: **for** $k = 0, 1, 2, \dots$ **do**
3: Collecter un ensemble de trajectoires $\mathcal{D}_k = \{\tau_i\}$
en appliquant la politique $\pi_k = \pi(\theta_k)$ dans l'environment.
4: **for** $t = 0$ to $t = T$ **do**
5:
$$\theta_{k+1} = \theta_k + \alpha \nabla_{\theta_k} \log \pi_{\theta_k}(a_t|s_t)R(\tau)$$
6: **end for**
7: **end for**

3.4.3 Acteur - Critique

L'acteur - critique est une approche hybride entre le calcul de la fonction de valeur et l'optimisation de la politique. Le calcul de la fonction de valeur aide à l'optimisation de la politique.

One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

FIGURE 6: Exemple d'acteur - critique d'après Sutton and Barto [19]

L'acteur - critique présenté sur la figure 6 montre la dépendance de l'acteur vis-à-vis du critique. A la ligne 13, la mise-à-jour du paramètre θ paramétrant la politique, invoque δ qui est mis-à-jour avec la fonction de valeur. On voit l'intéraction entre le critique et l'acteur. Nous avons un système de rétroactions positives, en améliorant le critique, on améliore l'acteur qui prendra de meilleures décisions et augmentera la valeur des états parcourus par le critique.

Nous avons vu les bases des réseaux de neurones et de l'apprentissage par renforcement. Nous étudions leur application dans le cas de la navigation dans la section suivante.

3.5 Robotique mobile et apprentissage par imitation

En 2005, Muller et al. [11] utilise un réseau de neurones pour construire un système autonome de bout-en-bout. Un robot mobile est entraîné à éviter des obstacles en navigation tout-terrain. L'entrée est constituée d'une image à six canaux de dimension 149 x 58 x 6 produite par deux cameras RGB en stéréo. Nous avons, par ailleurs, un réseau de neurones convolutif contenant 72000 paramètres qui est entraîné.

L'entraînement est effectué de manière supervisée. Ils ont collecté de nombreuses missions humaines sur une large variété de terrains en enregistrant les images données par les caméras stéréo et les commandes données par le pilote. Les deux commandes sont discrètes : tourne-à-droite et tourne-à-gauche.

A la fin de l'entraînement l'écart entre les commandes données par le pilote et les commandes inférées par le réseau de neurones convolutif doit être le plus faible possible comme dans un problème classique de régression. Avec cette approche, le robot peut très bien détecter les obstacles et les contourner à une vitesse d'environ 2 m/s. On appelle cette technique : l'apprentissage par imitation.

Cette technique est encore utilisée en 2017 avec Rausch et al. [13]. L'entrée est une image RGB, la sortie est uniquement un angle de direction mais continu cette fois. La technique s'améliore grâce à des jeux de données important et des capacités de calculs bien plus élevées. Le réseau de neurones convolutif utilisé est par ailleurs complexifié.

Afin de tenir compte de l'aspect séquentiel des entrées et des actions. Xu et al. [22] a ajouté une cellule LSTM (Long Short Term Memory). Les LSTM sont des réseaux de neurones récurrents, ils tiennent compte des états passés. L'entrée de la cellule LSTM sera la sortie du réseau de neurones convolutif concaténée à l'action précédemment choisie. De cette manière, nous pouvons tenir compte des motifs et des séquences. La navigation est plus robuste par rapport à un réseau de neurones convolutif seul. En effet, l'environnement ne vérifie pas totalement l'hypothèse de Markov selon laquelle, il suffit de connaître l'état présent pour prédire le futur.

En résumé, l'apprentissage par imitation nécessite des bases de données d'entraînement intégrant les décisions temps réel du pilote qui sont peu répandues donc cela limite la capacité de généralisation de ces méthodes.

3.6 Robotique mobile et apprentissage par renforcement

L'apprentissage par renforcement a fait ses preuves dans le domaine des jeux-videos. Mnih et al. [9] ont montré qu'un algorithme de Deep-Q-Learning (basé sur la fonction de valeur) surpasse toutes les IA développées et peut être adapté à un jeu quelconque sans modifier la structure de l'algorithme. L'apprentissage par renforcement a également de plus en plus d'applications dans le domaine de la robotique mobile. Nous pouvons distinguer deux types d'entrées : LIDAR et image.

3.6.1 Apprentissage par renforcement avec données LIDAR pour de la navigation

Surmann et al. [18] contrôle un robot mobile de bout-en-bout en utilisant un algorithme de renforcement de type acteur-critique asynchrone basé sur la fonction d'avantage. L'entrée de l'algorithme (7) est un nuage de points rendant compte la 3D de l'environnement et provenant de la fusion entre une carte de profondeur et d'une nappe de LIDAR 2D. Nous avons aussi comme entrée, l'orientation de l'agent par rapport à la cible encodée en *one-hot* (un nombre entre 1 et 3 est codé par un vecteur de 0 et de 1, dans cet encodage 1=[1, 0, 0], 2=[0, 1, 0] etc.). Les actions données par l'acteur sont discrètes. L'agent reçoit une récompense positive si la cible est atteinte ou s'il se rapproche en distance et en angle, en revanche il reçoit une pénalité si un obstacle est touché ou que la mission ne se termine pas, i.e. le nombre de transitions limite est dépassé. L'algorithme est entraîné sur un simulateur cinématique simplifié et rapide. Après 1h d'entraînement sur un unique GPU, les épisodes se terminent avec un taux de succès de 3/4. Les mêmes entrées et la même fonction de récompense ont été choisies par Marchesini and Farinelli [8].

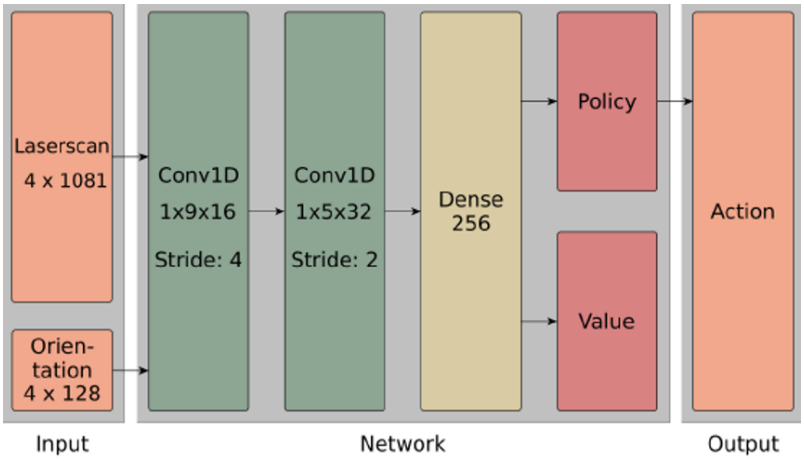


FIGURE 7: Architecture du réseau de neurones Surmann et al. [18]

3.6.2 Apprentissage par renforcement avec données visuelles pour de la navigation

Wijmans et al. [20] ont atteint un bon niveau de performance dans des environnements simulés intérieurs. L'algorithme utilisé est le proximal policy optimisation (PPO). Il est considéré comme l'état de l'art car performant et stable. PPO est un algorithme de type acteur-critique *on-policy* (les trajectoires sont collectées avec la politique la plus récente). Sur la dataset Gibson (Xia et al. [21]), ils atteignent 90 / 100 de taux de succès en utilisant comme entrée la position polaire de la cible dans le répère du robot ainsi qu'une carte de profondeur. Le résultat est amélioré en ajoutant les canaux RGB.

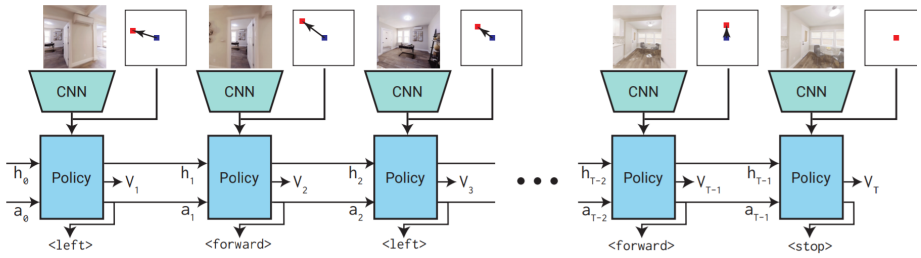


FIGURE 8: Architecture du réseau de neurones d'après Wijmans et al. [20]

Les entrées RGB et la carte de profondeur sont encodées par un réseau de neurones de convolution : un backbone CNN de type ResNet34 (figure 8). La position polaire de la cible est concaténée à l'encodage. Le bloc politique contient à la fois le réseau de neurones récurrent (RNN) utilisant des cellules GRU (Chung et al. [2]) et l'acteur-critique propre à l'algorithme PPO (Schulman et al. [16]).

Kulhánek et al. [7] ont aussi développé un réseau de type acteur-critique avec en entrée une image RGB (figure 9). Cette image est encodée en un espace latent par un réseau de neurones convolutif et par un LSTM (réseau de neurones récurrent c.f. Staudemeyer and Morris [17]). L'espace latent est l'entrée d'un acteur-critique utilisant la fonction avantage pour définir le critique (algorithme A2C Mnih et al. [10]). La spécificité de cette étude est d'utiliser des tâches auxiliaires pour stabiliser

et améliorer la convergence du réseau de neurones convolutif (figure 9). Ces tâches sont des auto-encodeurs : une technique classique de deep-learning quand nous n'avons pas assez de données (cf. chapitre 14 de Goodfellow et al. [3]). Ils obtiennent de bon résultats.

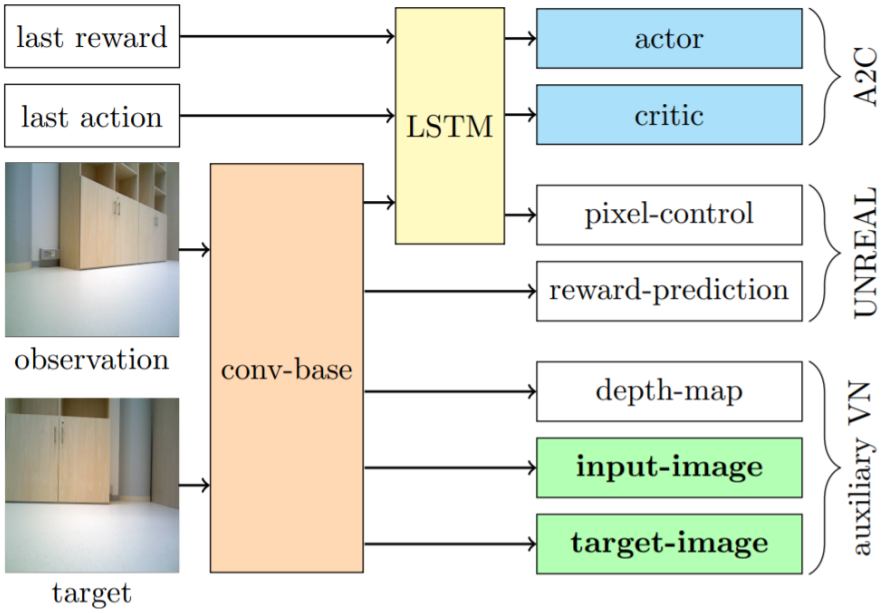


FIGURE 9: Architecture du réseau de neurones d'après Kulhánek et al. [7]

3.7 Multi-modalité, perception de l'environnement et transfert

La perception de l'environnement est importante avec l'approche modulaire. Elle l'est aussi pour définir l'entrée de l'algorithme de renforcement, elle améliore la compréhension de la scène pour le choix des descripteurs.

3.7.1 Multi-modalité

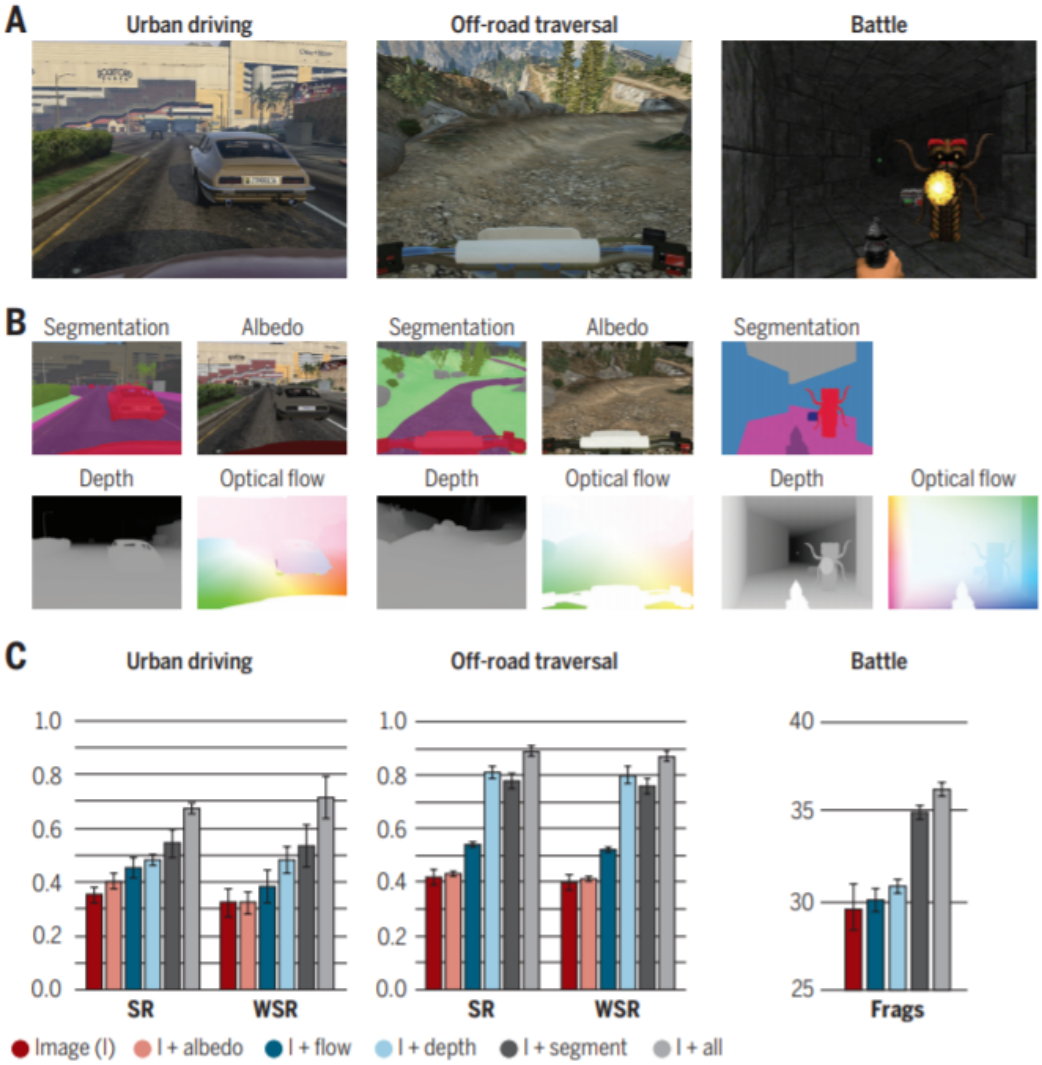


FIGURE 10: Impact de la modalité choisie sur la performance de l'algorithme d'après Zhou et al. [24]

Zhou et al. [24] ont montré qu'utiliser un module de perception en plus pour améliorer l'image augmente significativement le taux de succès. Nous pouvons voir sur la figure 10 que dans un environnement *off-road*, ajouter une carte de profondeur à l'image RGB seule double le taux de succès.

3.7.2 Perception de l'environnement et transfert

Le transfert du réseau de neurones entraîné en simulation dans le monde réel est difficile pour plusieurs raisons. Les capteurs ne sont pas parfaitement modélisés dans le simulateurs (incertitudes et bruits), la lumière réelle peut modifier la perception de l'environnement comme par exemple un reflet sur une surface humide ou métallique. Enfin la dynamique du véhicule est toujours différente

lors du transfert. Nous pouvons par exemple avoir un glissement des roues lors du démarrage suivant la surface au sol, l'état des pneus du robot et la commande envoyée. L'apprentissage dans le monde réel est difficile à mettre en place. Nous souhaitons un transfert sans ajustement (*fine-tuning*) et l'apprentissage de bout-en-bout est un obstacle à cela. Pour éviter ce problème, Müller et al. [12] entraînent leur robot en simulation seulement sur des images déjà segmentées par un encodeur sémantique performant entraîné sur un jeu de données d'images. La politique de navigation est apprise sur simulateur mais au lieu d'avoir comme actions des commandes de vitesse et de direction, la politique produit des waypoints. Le régulateur PID du robot le dirige vers ces waypoints. De cette manière, l'apprentissage ne dépend plus de la dynamique du véhicule. Les auteurs de l'article ont montré que leur approche surpasse l'état de l'art au niveau de la généralisation dans des mondes simulés et réels.

3.8 Synthèse de l'état de l'art

A la lumière de cet état de l'art, nous nous concentrerons dans notre méthodologie sur les points suivant.

3.8.1 Espace d'état

Avec la fonction de récompense, les observations contenues dans l'espace d'état sont les seules informations qu'on a sur l'environnement. Il est donc important de bien choisir cet espace d'état.

Une navigation performante dans des environnements les plus complexes a été obtenue avec des images en entrée. Néanmoins utiliser une image améliorée (sémantique, profondeur) permet d'obtenir de meilleurs résultats (d'après Zhou et al. [24]). De plus, en comparaison avec une entrée LIDAR, le temps d'apprentissage est bien plus important avec une image. L'image impose d'entraîner un réseau de neurones convolutif coûteux en temps de calcul.

Une localisation précise du robot semble nécessaire si la cible est une coordonnée dans un repère : *point-goal navigation*.

Afin de pouvoir évoluer dans des environnements intérieurs complexes, nous allons utiliser une image ou une image augmentée (profondeur, sémantique) en entrée de l'algorithme en plus de la position polaire.

3.8.2 Fonction de récompense

La fonction de récompense est notre unique moyen de faire comprendre la tâche à notre agent. Il faut la construire comme une somme pondérée de termes. Chaque terme correspond à une contrainte que l'on fixe sur le robot. Les poids sont quantifiés proportionnellement à leur importance dans la tâche.

3.8.3 Espace d'actions

L'espace d'actions peut être continu ou discret.

Pour la navigation de robots mobiles utilisant des méthodes de renforcement, les résultats les plus probants ont été obtenus avec des actions discrètes. En choisissant des actions discrètes, on a un module de contrôle interprétable séparé du réseau de neurones. Le transfert d'un environnement cinématique à un environnement dynamique est plus facile car le contrôle sur la dynamique est en dehors du réseau de neurones. De cette manière, vis-à-vis du réseau de neurones l'environnement dynamique et l'environnement cinématique sont les mêmes.

Une navigation fluide du robot nécessite une action choisie dans un intervalle. Choisir une action continue, contraint le choix de l'algorithme.

Nous souhaitons une navigation à actions continues afin d'avoir une trajectoire plus fluide, précise et économe.

3.8.4 Algorithme

Les algorithmes de type acteur-critique donnent les meilleurs résultats. Ils prennent en entrée un espace de descripteurs latent commun à l'acteur et au critique. L'espace latent est l'encodage d'une image par un réseau de neurones convolutif. Pour tenir compte de l'aspect non markovien de l'environnement, nous pouvons utiliser un réseau de neurones récurrent en sortie du réseau de neurones convolutif.

Si l'espace d'actions est continu, il faut choisir un algorithme ayant une politique stochastique : le proximal policy optimisation (PPO) en est un.

3.8.5 Simulateur

L'apprentissage en ligne est possible pour des tâches nécessitant un nombre faible d'étapes d'entraînement. En considérant notre tâche de navigation et au vu de l'état de l'art, avant d'obtenir une convergence de l'algorithme, il faut effectuer des dizaines de millions de transitions. Ce qui correspond à des dizaine de milliers d'épisodes de 500 transitions. Il est donc, pour nous, nécessaire d'utiliser un simulateur.

Des simulateurs simples et performants en temps de calculs sont utilisés au début de l'entraînement, ce sont des simulateurs cinématiques. Même si l'environnement simulé ne rend pas parfaitement compte de la tâche visée et de toutes les interactions, une première convergence des poids du réseau de neurones est obtenue. Un simulateur plus réaliste et dynamique est ensuite envisagé.

4 Algorithme et méthode d'entraînement utilisés

Dans cette section nous détaillerons l'algorithme puis son architecture. Enfin nous présenterons l'implémentation et la méthode d'entraînement.

4.1 Proximal Policy Optimisation(PPO)

Dans cette section nous allons étudier l'algorithme définissant l'état de l'art des algorithmes basés sur l'optimisation de la politique : Proximal Policy Optimization (PPO). PPO est un algorithme de type acteur-critique, comme REINFORCE, il optimise directement la politique mais effectue aussi une estimation de la fonction de valeur comme dans Q - Learning.

PPO est un algorithme stable que nous avons choisi d'utiliser dans nos travaux

Comme dans REINFORCE, nous voulons maximiser une fonction objectif $L^{\text{CLIP}}(s, a, \theta_k, \theta)$ par ascension de gradient par exemple :

$$\theta_{k+1} = \arg\max_{\theta} \underset{s,a\sim\pi_{\theta_k}}{\mathrm{E}} \left[L^{\text{CLIP}}(s, a, \theta_k, \theta) \right]$$

On a $L^{\text{CLIP}}(s, a, \theta_k, \theta)$ qui vaut l'expression suivante :

$$L^{\text{CLIP}}(s, a, \theta_k, \theta) = \min \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \quad \text{clip} \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1 - \epsilon, 1 + \epsilon \right) A^{\pi_{\theta_k}}(s, a) \right)$$

Le quotient $r = \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$ est le taux de changement entre l'ancienne politique $\pi_{\theta_k}(a|s)$ et la nouvelle politique $\pi_\theta(a|s)$ dans la probabilité de choisir une action a, si r est grand alors on a confiance en notre action. Inversement si r est faible alors on a moins de certitudes. $A^{\pi_{\theta_k}}(s, a)$ est la fonction avantage : $A^{\pi_{\theta_k}}(s, a) = V(s) - Q(s, a)$ elle estime la valeur d'une action a prise dans l'état s. ϵ est le paramètre de clipping, il assure un changement pas trop important de la politique.

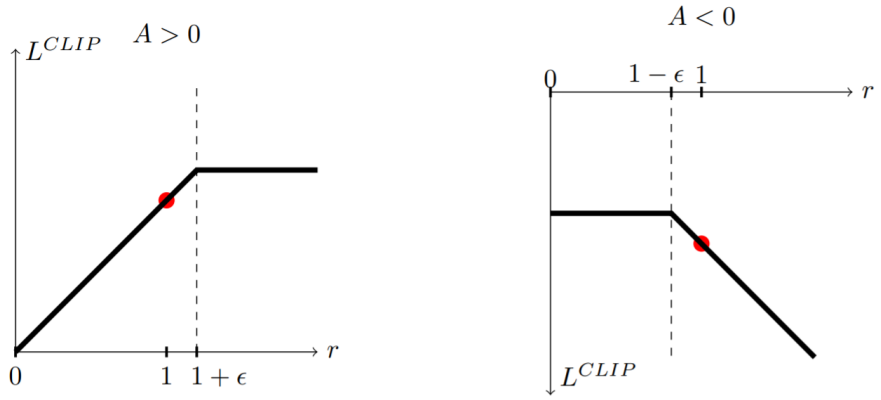


FIGURE 11: Le graph de la fonction objectif, le point rouge correspond à la position initiale de la fonction objectif, $r = \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$ d'après Schulman et al. [16]

Si $A > 0$ (figure 11 gauche) alors cela signifie que la nouvelle action estimée par la politique est juste car elle permet d'améliorer la valeur de l'état. La fonction objectif doit alors augmenter proportionnellement à la confiance que l'on a en notre action. Toutefois afin d'assurer une stabilité de la politique, on souhaite borner l'évolution de la fonction objectif dans le cas d'un r grand, ce qui correspond à un excès de confiance. Si r est faible (défiance) alors la politique n'estime pas l'action à sa juste valeur. Il est normal de moins valoriser l'action.

Si $A < 0$ (figure 11 droite) alors cela signifie que la nouvelle action estimée par la politique n'est pas satisfaisante car elle ne permet d'améliorer la valeur de l'état. La fonction objectif doit alors diminuer proportionnellement à la confiance que l'on a en notre action. En effet si r est grand, on a confiance en notre action ce qui est dangereux. Afin d'assurer une stabilité de la politique, on souhaite borner l'évolution de la fonction objectif dans le cas d'un r faible.

Une valeur raisonnable de ϵ est 0.2 donc dans le cas d'un bon comportement la fonction objectif varie au plus de 20%. Ce mécanisme assure à l'algorithme de ne pas trop s'écarter de la politique : "avancer lentement mais surement".

Algorithme 2 : Algorithme Proximal Policy Optimization (PPO) d'après Schulman et al. [16]

1: Entrées : paramètres initiaux de la politique θ_0, paramètres initiaux de la fonction de valeur ϕ_0
2: **for** $k = 0, 1, 2, ...$ **do**
3: Collecter un ensemble de trajectoires $\mathcal{D}_k = \{\tau_i\}$ en appliquant la politique $\pi_k = \pi(\theta_k)$ dans l'environment.
4: Calculer les sommes de récompenses sur la trajectoire k à partir de chaque instant t $\hat{R}_t$.
5: Estimer les avantages à chaque instant t, $\hat{A}_t$ en se basant sur la fonction de valeur V_{ϕ_k}.
6: Mettre à jour la politique en maximisant la fonction objectif :

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

en utilisant par exemple l'ascension stochastique de gradient d'Adam.
7: Ajuster la fonction de valeur par regression sur l'erreur des moindres-carrés :

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

en utilisant par exemple un algorithme de descente de gradient.
8: **end for**

4.2 Architecture de l'algorithme

Afin d'avoir la meilleure compréhension de la scène et de son environnement. Nous choisissons comme entrée de l'algorithme une carte de profondeur ainsi que la position en coordonnées polaires

de la cible dans le repère de l'agent.

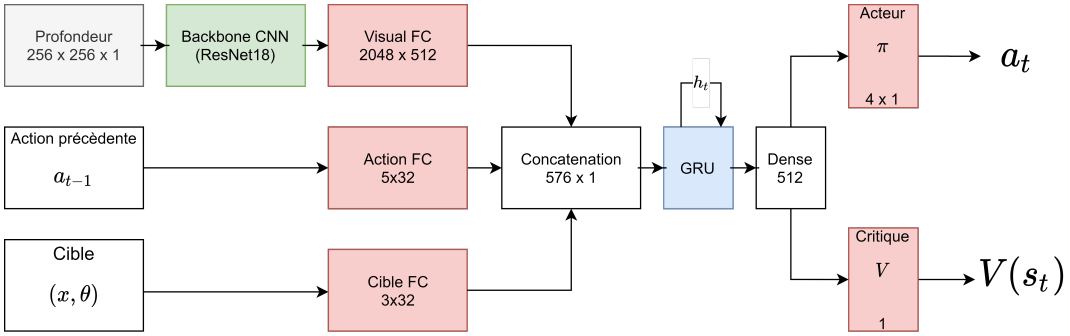


FIGURE 12: Architecture globale du réseau de neurones utilisé pour notre algorithme PPO

La carte de profondeur (tenseur de dimension 256 x 256 x 1 contenant des valeurs normalisées dans l'intervalle [0, 1]) est encodée par un réseau de neurones convolutif. Pour limiter le temps d'entraînement et permettre d'embarquer l'algorithme sur une plateforme réelle, nous avons identifier les architectures des réseaux de neurones convolutifs suivantes : ResNet18 et MobileNet. Ces architectures ont le moins de paramètres à entraîner (moins de 10 M) , nécessitent le moins d'opérations et ont des bons niveaux de performance sur les problèmes de vision comme le montre Bianco et al. [1] (figure 13). Le réseau de neurones convolutif permet d'extraire les bons descripteurs visuels en diminuant fortement la dimension de l'entrée passant de 65536 (256 x 256 x 1) à 2048. Ensuite une couche *fully-connected* réduit la dimension de ce tenseur à 512. Ce tenseur est stocké dans une liste pour être ensuite concaténé aux autres entrées. L'action précédente ainsi que la position de la cible en coordonnées polaires sont transformées par une couche *fully-connected* et ajoutées à la liste contenant les données visuelles. Les tenseurs présents dans la liste sont concaténés, nous obtenons alors un tenseur de dimension 576 (figure 12).

Ce tenseur d'état est l'une des entrées d'une cellule récurrente de type GRU (Gated Recurrent Unit) car la navigation ne vérifie pas totalement la propriété de Markov (le futur ne dépend pas des états passés mais seulement de l'état présent). A la sortie de la cellule récurrente, nous avons un tenseur de dimension 512 qui constitue un espace latent qui est l'entrée de l'acteur et du critique propre à l'algorithme PPO décrit précédemment.

Le critique donne la valeur d'un état et l'acteur indique une action représentée par une clé dans le dictionnaire suivant : 0 : STOP, 1 : AVANCE (10 cm), 2 : TOURNE-A-GAUCHE (-10 degrès), 3 : TOURNE-A-DROITE (+10 degrès)

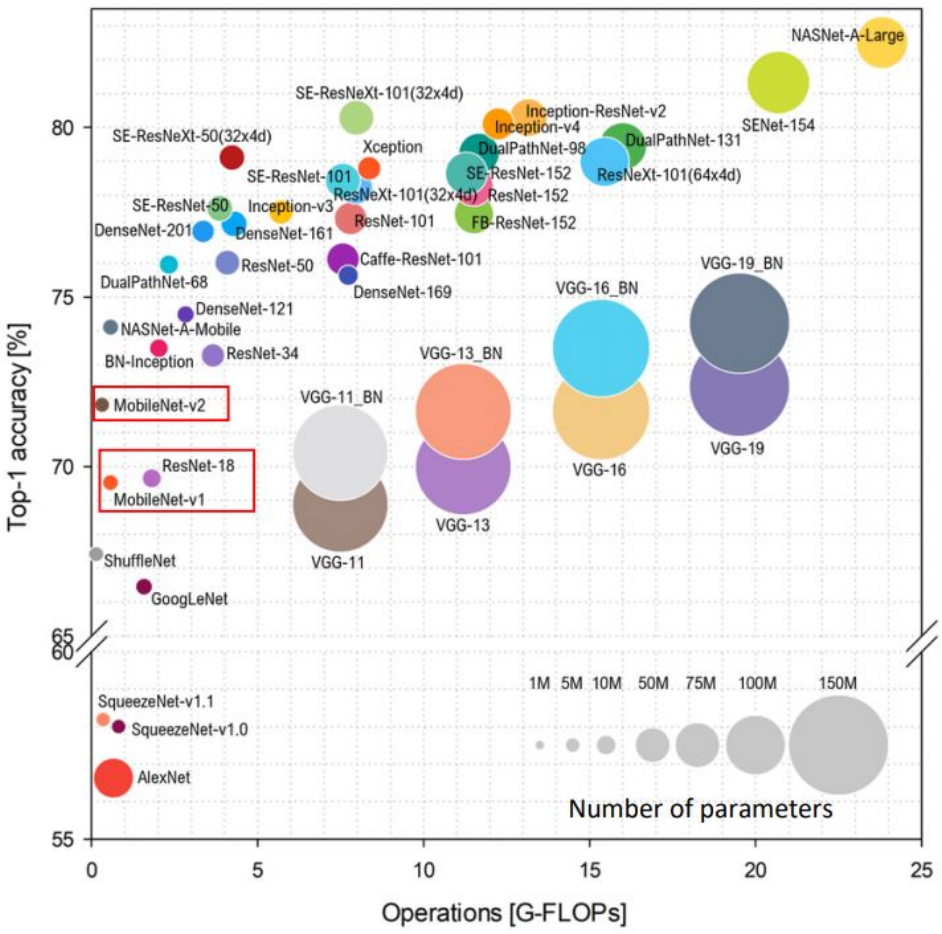


FIGURE 13: Benchmark des architectures des réseaux de neurones convolutifs d'après Bianco et al. [1]

4.3 Méthode d'entraînement

Dans notre cas, l'apprentissage par renforcement, nécessite un entraînement préalable sur simulateur. En effet, entraîner directement l'algorithme sur le robot réel n'est pas envisageable pour plusieurs raisons : au début de l'entraînement la politique est aléatoire, le robot est alors en phase d'exploration. Il peut alors percuter des obstacles, être bloquer dans des endroits de l'espace sans issues possibles. Il faudrait alors manuellement le replacer à un endroit de la salle et ainsi prendre plus de temps par épisode. D'autre part, de manière générale dans le domaine du machine-learning, un algorithme performant repose sur un entraînement varié. Durant l'entraînement, le réseau de neurones estimant la politique doit avoir été confronté au plus large spectre de scènes possible. Plus ce spectre est étendu, plus le robot sera capable de généraliser et ainsi proposer les bonnes actions dans de nombreuses situations. Pour ces raisons entrainer *from scratch* le robot en ligne est impossible, dans notre problème.

Nous devons donc choisir le meilleur simulateur adapté à notre problème. Ce simulateur doit posséder plusieurs propriétés. D'abord le simulateur doit être rapide afin d'effectuer le plus d'étapes possibles dans l'environnement et ainsi entraîner le robot dans un temps raisonnable. Nous pouvons,

dès lors exclure un simulateur reposant sur un moteur physique comme Gazebo ou Unity. En effet, bien que réalistes ces simulateurs physiques sont réputés lents. Le simulateur doit aussi permettre de faire évoluer le robot dans un large spectre d'environnements intérieurs, on doit pouvoir intégrer des capteurs sur le robot : au moins une caméra de la profondeur ou une camera RGB, et un positionnement spatial.

4.3.1 Simulateurs

Le simulateur est important pour permettre une première convergence des poids du réseau de neurones. Il existe de nombreux simulateurs avec différent niveaux de réalisme et de rapidité. Pour valider une première implémentation, il est intéressant de la tester sur un simulateur rapide.

Gym Gym est un simulateur développé par OpenAI [2], il propose différents environnements de simulations. Le plus classique est Cart-Pole, il s'agit d'un pendule inversé décrit par Sutton and Barto [19].

Le formalisme de l'apprentissage par renforcement est intégré au simulateur.
L'instruction `env.reset()` retourne une observation qui sera notre vecteur d'état : position sur l'axe, vitesse sur l'axe, angle de la tige, vitesse angulaire de la tige. Une inférence de ce vecteur d'état sur le réseau de neurones retourne une action discrète : déplacement à droite ou déplacement à gauche. L'instruction `env.step(action)` met à jour l'observation suite à l'action effectuée par l'agent. L'agent obtient une récompense de +1 si l'action n'a pas mené à un état terminal. Un état terminal est obtenu dans plusieurs cas de figure : soit la position du cart est supérieur à 2.4 m en valeur absolue par rapport à l'origine de l'axe, soit l'angle de la tige est supérieur à 12 degrés, soit le nombre d'étapes effectuées est supérieur à 500 (étapes effectuées avec succès). Cette fonction de récompense encourage l'agent à maximiser le nombre d'étapes par épisode et donc que la tige soit la plus verticale possible et que l'agent soit centré sur l'origine de l'axe.

Avec une bonne implémentation et un ordinateur portable (sans GPU et un CPU), en une vingtaine de minutes, la fonction de récompense converge vers 500 points de récompense par épisode. Dans ce cas, l'implémentation peut fonctionner sur le problème de navigation, la condition est nécessaire mais pas suffisante.

En résumé, Gym peut être un premier test d'implémentation car il propose des environnements performants. Afin de valider un algorithme de renforcement pour de la navigation, Gym ne suffit pas. En effet, aucun des environnements proposés par Gym ne décrit un problème de navigation en intérieur.

Habitat Savva et al. [15] ont proposé un environnent de simulation pour de la navigation en intérieur performant. Cet environnement a été développé par Facebook AI Research. Ils ont organisé un challenge pour entreprendre des recherches et tester des algorithmes. La compétition se divise en deux challenges distincts : point-goal navigation et object-goal navigation. Le challenge qui se

2. https://gym.openai.com/

rapproche le plus de notre problème de navigation est point-goal navigation. Ce challenge intégralement virtuel et étendu sur une année départage les candidats sur leur capacité à faire naviguer leur robot dans l'environnement Habitat en minimisant différentes métriques comme par exemple : la distance parcourue, le nombre d'obstacles touchés. Habitat intègre différentes couches (figure 14).

Habitat Sim La première couche est Habitat-Sim : il s'agit d'un simulateur 3D d'environnements intérieurs avec des agents configurables, les dimensions de l'agent peuvent par exemple être modifiées et il peut embarquer différents capteurs. L'environnement repose sur le rendu en 3D de datasets générique d'environnement intérieur comme par exemple les datasets Matterplot3D ou Gibson (Xia et al. [21]). Nous avons fait le choix d'utiliser la dataset Gibson pour sa rapidité de chargement et les performances obtenues par les agents dans le challenge Habitat. Gibson représente 572 bâtiments, 1477 étages et 211 km^2 de surface navigable. Ce large spectre d'environnements navigables garantit une bonne généralisation de l'algorithme de renforcement en limitant le surapprentissage sur un nombre limité de scènes. La bonne capacité de généralisation est un point essentiel au transfert de l'algorithme sur robot réel. En performance, Habitat-Sim est optimisé, Habitat-Sim peut fonctionner en théorie sur un unique GPU à 10 000 images par seconde. La rapidité de convergence d'un algorithme de renforcement dépend en partie de la capacité à générer rapidement des jeux de données.

Habitat API La seconde couche reposant sur Habitat-Sim est Habitat - API. La tâche à effectuer par le robot est définie dans Habitat-API. La définition d'une tâche consiste à générer dictionnaire au format *. json, où nous définissons les épisodes pour notre apprentissage : la scène à charger, la position initiale du robot, la position de la cible dans l'environnement et éventuellement le plus court chemin (pour pouvoir estimer la qualité d'un épisode). Un deuxième fichier de configuration pointe vers le dictionnaire et spécifie les dimensions de l'agent, les capteurs utilisés, la durée maximale d'un épisode, la politique à utiliser et de nombreux autre paramètres utiles à l'entraînement. Une fois l'environnement chargé avec Habitat – API, nous pouvons faire naviguer l'agent dans l'environnement avec les mêmes instructions que dans Gym. Le passage de Gym à Habitat – API est ainsi facilité. Un module permet de visualiser des inférences au cours de l'entraînement.

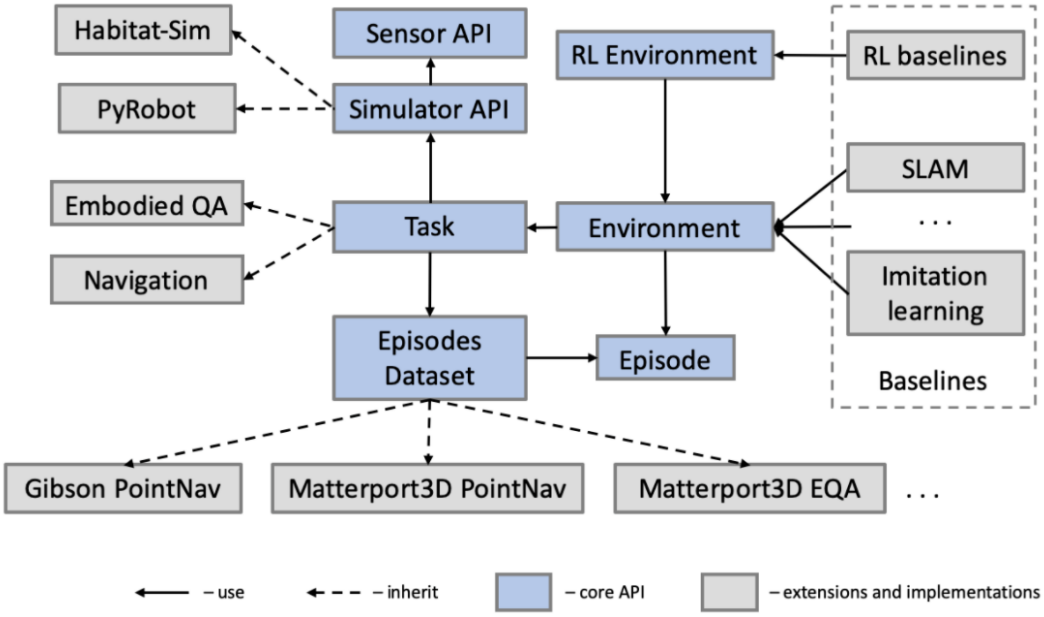


FIGURE 14: Architecture du simulateur Habitat d'après Savva et al. [15]

FIGURE 15: Données visuelles présentes dans la dataset Gibson d'après Xia et al. [21]

FIGURE 16: Définition d'un épisode de navigation dans Habitat d'après Savva et al. [15]

Habitat est un bon candidat pour effectuer le premier entraînement de notre réseau de neurones car il est rapide et intègre des scènes réalistes et variées.

Gazebo Gazebo[3] est un simulateur dynamique 3D capable de simuler avec précision et efficacité des populations de robots dans des environnements intérieurs et extérieurs complexes. Bien que similaire aux moteurs de jeux, Gazebo offre une simulation physique à un degré de fidélité beaucoup plus élevé, une série de capteurs et des interfaces pour les utilisateurs et les programmes.

3. http://gazebosim.org/

Utiliser Gazebo permet de valider l'archictecture logicielle sous ROS et le comportement dynamique du robot avant de passer sur le monde réel.

Pipeline de l'entrainement à l'inférence sur robot Comme le montre la figure 17, nous entraînerons d'abord notre algorithme sur Habitat qui est un simulateur cinématique rapide, puis nous validerons l'entraînement dans Habitat ainsi que dans le simulateur physique Gazebo avec une option possible de *fine-tuning* (réapprentissage) et enfin nous passerons dans le monde réel avec aussi une option possible de *fine-tuning*.

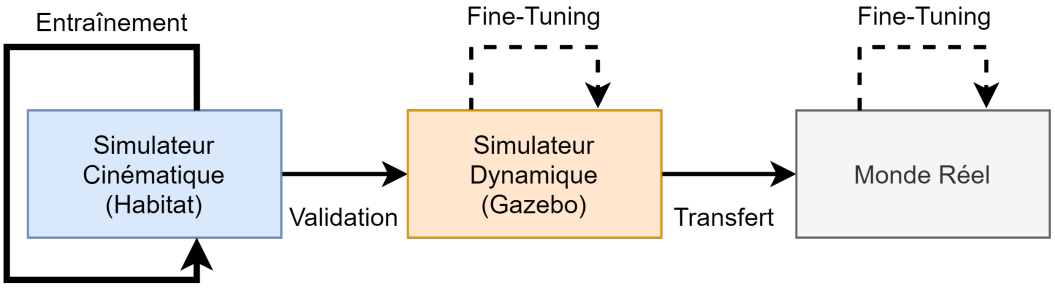


FIGURE 17: Méthode d'utilisation des simulateurs

4.3.2 Fonction de récompense

Nous devons définir l'intéraction entre l'agent et son environnement. La récompense est la seule information que l'agent obtient de l'environnement. Elle essentielle au bon fonctionnement de l'algorithme. Comme nous avons pu le voir dans l'état de l'art. Il y a une grande variété de récompenses. Nous avons choisi de tester plusieurs fonctions de la récompense. Elles sont des sommes pondérées de plusieurs termes représentant nos objectifs de navigation : le robot se dirige le plus rapidement possible vers la cible et en sécurité (il ne doit pas heurter d'obstacles sur son trajet). Nous proposons donc la fonction de récompense suivante :

$$r(s,a) = w_{sp} r_{sp} + \mathbf{1}_{suc} r_{suc} + w_d r_d + \mathbf{1}_{col} r_{col}$$

- $w_{sp} r_{sp}$ correspond au poids et à la pénalité associée pour influencer sur la vitesse de déplacement, c'est une constante initialisant chaque récompense immédiate (voir figure 18).
- $\mathbf{1}_{suc} r_{suc}$ correspond à la fonction indicatrice du succès, elle vaut 1 si l'épisode a terminé et 0 sinon et à la récompense associée au succès d'un épisode, on rappelle qu'un épisode termine si l'action "STOP" est donnée par la politique et si l'agent reçoit ce "STOP" à moins de 0.2 m de la cible.
- $w_d r_d$ correspond au poids associé à la récompense / pénalité pour influencer sur la distance, $r_d = d_{t-1} - d_t$. Si la distance en norme L_2 entre l'agent et la cible diminue entre deux instants alors la récompense est légèrement positive et réciproquement.
- $\mathbf{1}_{col} r_{col}$ correspond à la fonction indicatrice d'une collision, si il y a eu une collision alors elle vaut 1 et 0 sinon, r_{col} correspond à la pénalité associée à une collision.

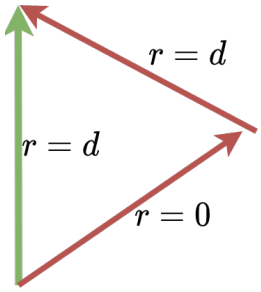


FIGURE 18: La pénalité r_{speed} permet d'éviter le comportement décrit par les flèches rouges. Si un épisode consiste à relier les sommets d'un triangle équilatéral et que la pénalité de vitesse est annulée alors la trajectoire rouge est deux fois plus longue mais la qualité de la trajectoire estimée par l'environnement par le biais de la fonction de récompense est la même. Avec une pénalité de vitesse, on privilégie la trajectoire directe (verte) et la trajectoire rouge ne sera pas choisie.

4.3.3 Implémentation de l'algorithme

Les algorithmes d'apprentissage par renforcement sont très sensibles à l'implémentation, nous avons choisi le *framework* d'apprentissage par renforcement développé par l'équipe de Facebook AI Research pour l'implémentation de l'algorithme PPO (Wijmans et al. [20]).

Le réseau de neurones est défini avec la librairie PyTorch. Du fait de PyTorch, la construction de l'arbre de calcul est dynamique. Lors de la phase d'inférence, les branches nécessitant le calcul d'un gradient (`requires_grad = True` dans PyTorch) sont stockées dans l'arbre de calcul de la racine du réseau de neurones constituée par les encodeurs, jusqu'aux feuilles du réseau de neurones constituées par l'acteur et le critique. A la fin d'une inférence, nous effectuons la rétro-propagation par l'instruction `loss.backward()`, cette opération calcule les gradients de chaque branche en remontant les branches du réseau de neurones grâce à la règle de la chaîne (Higham and Higham [5]). Cette opération libère la mémoire de l'arbre de calcul. Enfin nous pouvons ajuster les poids des branches en appelant `optimizer.step()`.

4.4 Résultats de l'entraînement dans Habitat

Nous présentons ici l'apprentissage sur le simulateur Habitat. Afin d'évaluer la convergence nous utilisons quelques métriques d'évalutation.

4.4.1 Métriques d'évaluation

Succès pondéré par la longueur du chemin (*Success weighted by Path Length* SPL) Cette métrique est intégrée au simulateur Habitat elle permet de rendre compte de la qualité de la trajectoire. La définition du SPL est la suivante :

$$SPL = \frac{1}{N}\sum_{i=1}^{N} S_i \frac{l_i}{\max(p_i, l_i)}$$

avec l_i la distance du plus court chemin entre le départ et la cible, p_i la longueur du chemin pris par l'agent pendant l'épisode, S_i la fonction indicatrice du succès d'un épisode.

Comme $p_i \geq l_i$ le SPL appartient à l'intervalle $]0, 1]$ La longueur l_i peut être par exemple trouvée par l'algorithme A* (Sariff and Buniyamin [14]).

Si p_i est proche de l_i alors quotient $\frac{l_i}{\max(p_i, l_i)}$ est proche de 1 à l'inverse si l'agent effectue des détours alors $p_i \gg l_i$ et le quotient est faible.

Le SPL est une moyenne calculée sur la dataset de validation. Un SPL proche de 1 témoigne d'un algorithme de très bonne qualité.

Pour comparer deux algorithmes nous devons utiliser la même dataset de validation.

Nombre de collisions Le nombre d'obstacle percutés en moyenne par épisode sur la dataset permet de valider le critère de sécurité.

Un très bon SPL (proche de 1) est corrélé à un nombre de collisions élevé et réciproquement. Il faut trouver un équilibre entre ces deux métrique d'évaluations.

Succès pondéré par le temps de parcours sur la trajectoire (Success weighted by Time Length STL) Sa définition est la suivante :

$$STL = \frac{1}{N} \sum_{i=1}^{N} S_i \frac{t_i}{\max(a_i, t_i)}$$

avec t_i le temps de parcours du plus court chemin entre le départ et la cible, a_i le temps de parcours pris par l'agent pendant l'épisode, S_i la fonction indicatrice du succès d'un épisode. On remarque qu'il s'agit de la même formule que pour le SPL mais la grandeur est le temps et pas une longueur comme dans le SPL.

4.4.2 Serveur utilisé pour l'entraînement

Nous avons entraîné nos algorithmes sur un serveur du CEA constitué de 32 noeuds. Tous les noeuds sont des bi-processeurs INTEL Skylake avec support AVX512, avec au moins 48 threads et 192Go de mémoire RAM pour le GPU et 72 threads pour le CPU, avec un disque dur local de 2To. Les GPU sont des NVIDIA P5000. Chaque entraînement utilise un seul GPU mais nous pouvons lancer plusieurs entraînements avec des paramètres différents en même temps.

4.4.3 Résultats et comparaisons des fonctions de la récompense

Rappelons la fonction de récompense que nous avons utilisé :

$$r(s, a) = w_{sp} r_{sp} + \mathbf{1}_{suc} r_{suc} + w_d r_d + \mathbf{1}_{col} r_{col}$$

- $w_{sp} r_{sp}$ correspond au poids et à la pénalité associée pour influencer sur la vitesse de déplacement, c'est une constante initialisant chaque récompense immédiate (voir figure 18).
- $\mathbf{1}_{suc} r_{suc}$ correspond à la fonction indicatrice du succès, elle vaut 1 si l'épisode a terminé et 0 sinon et à la récompense associée au succès d'un épisode, on rappelle qu'un épisode termine si l'action "STOP" est donnée par la politique et si l'agent reçoit ce "STOP" à moins de 0.2 m de la cible.

- $w_d r_d$ correspond au poids associé à la récompense / pénalité pour influencer sur la distance, $r_d = d_{t-1} - d_t$. Si la distance en norme L_2 entre l'agent et la cible diminue entre deux instants alors la récompense est légèrement positive et réciproquement.
- $\mathbf{1}_{col} r_{col}$ correspond à la fonction indicatrice d'une collision, si il y a eu une collision alors elle vaut 1 et 0 sinon, r_{col} correspond à la pénalité associée à une collision.

Le terme de collision n'est pas défini dans l'état de l'art. Pour la suite, définissons les fonctions de la récompense suivante :

- OURS : $r(s,a) = w_{sp} r_{sp} + \mathbf{1}_{suc} r_{suc} + w_d r_d + \mathbf{1}_{col} r_{col}$
- SOA : $r(s,a) = w_{sp} r_{sp} + \mathbf{1}_{suc} r_{suc} + w_d r_d$

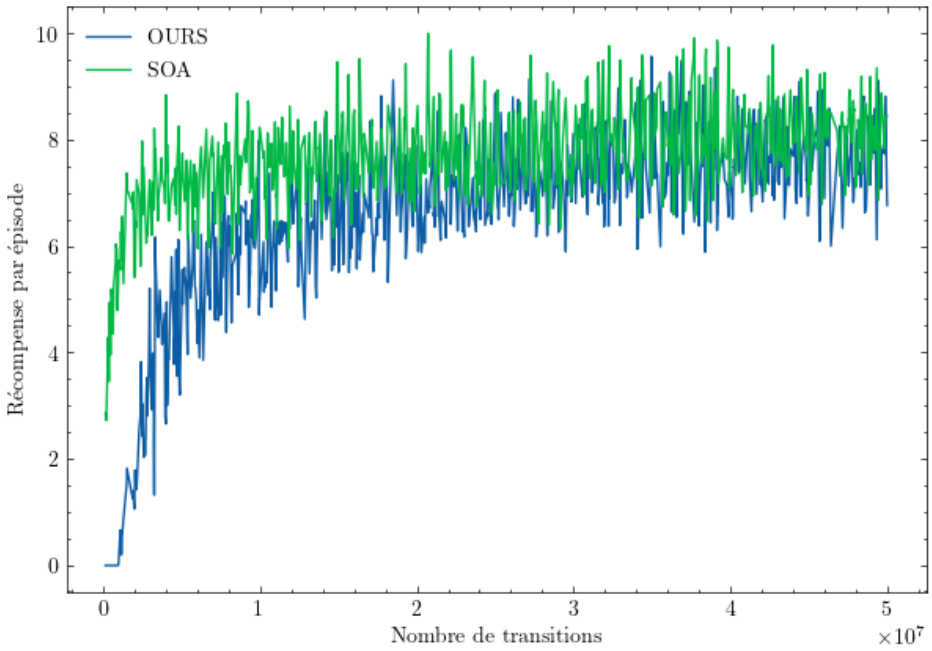


FIGURE 19: Récompense sur un épisode en fonction du nombre de transitions. Comparaison des entraînements sur 50 M de transitions avec deux fonctions de récompense différentes : OURS et SOA.

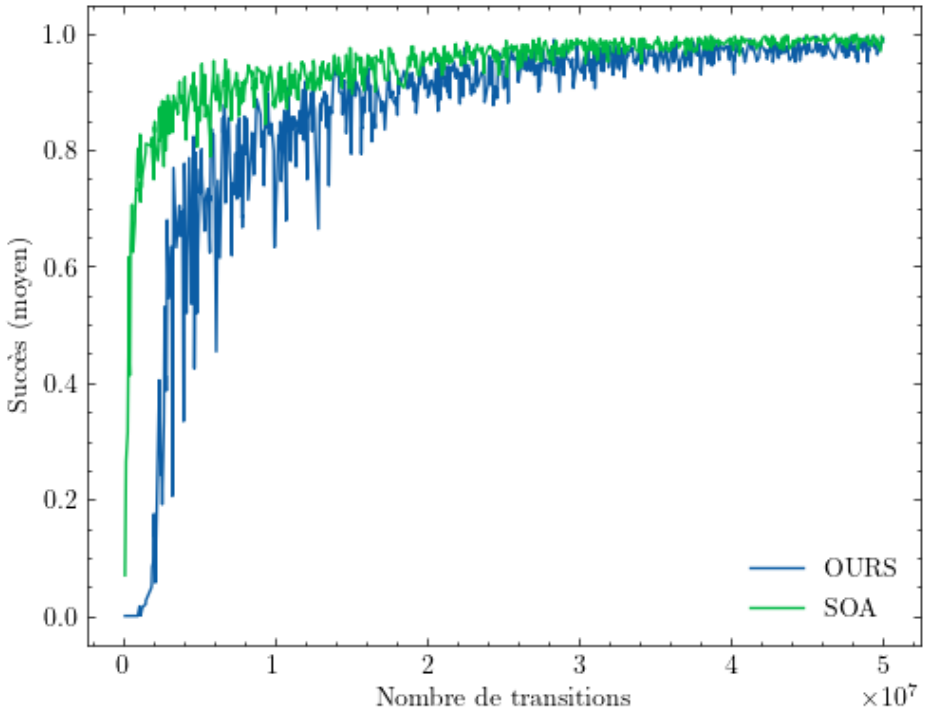


FIGURE 20: Succès moyen en fonction du nombre de transitions. Comparaison des entraînements sur 50 M de transitions avec deux fonctions de récompense différentes : OURS et SOA.

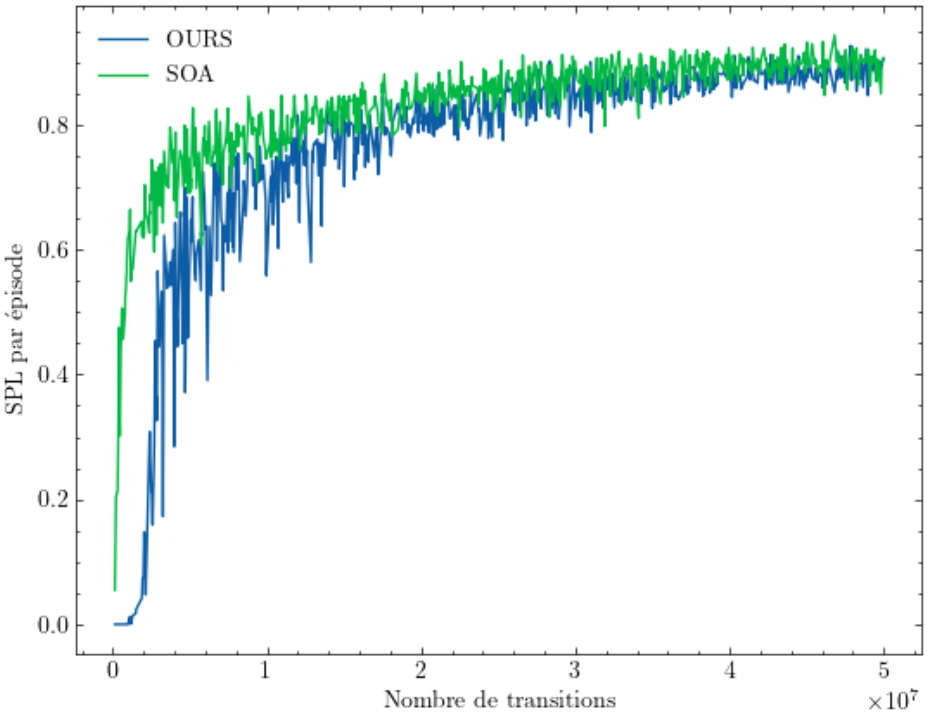


FIGURE 21: SPL en fonction du nombre de transitions. Comparaison des entraînements sur 50 M de transitions avec deux fonctions de récompense différentes : OURS et SOA.

Nous pouvons remarquer que toutes les courbes (figures 21, 20 et 19) sont corrélées les algorithmes ont convergé après environ 20 millions d'étapes. Toutefois la fonction de récompense tenant uniquement de la distance converge plus rapidement. En effet, au début de l'entraînement le comportement du robot est erratique, l'agent percute beaucoup d'obstacles et accumule les pénalités, il doit donc à la fois tenir compte de la tâche consistant à éviter les obstacles puis se rapprocher de la cible. Ce qui justifie ce retard. Néanmoins les deux fonctions algorithmes convergent vers des niveaux de performance équivalents. Nous pouvons noter que sur notre serveur et avec les hyperparamètres utilisés (voir la table 4), 50 M de transitions et 390 625 mises-à-jour des poids ont nécessité 55 heures d'entraînement.

TABLE 1: Influence du paramètre de collision sur le critère de sécurité, l'évaluation est effectuée sur une même dataset de 1000 épisodes variés

Comparaisons fonctions de la récompense en inférence sur une même dataset				
Fonction de récompense	Récompense moyenne	SPL (moyen)	Taux de succès (moyen)	Nombre de collisions (moyen)
SOA	7.05	0.86	0.96	18.4
OURS	5.22	0.81	0.92	1.66

Nous pouvons voir sur la table 1 que le terme de collision dans OURS diminue drastiquement le nombre de collisions sur un épisode et valide le critère de sécurité sans pour autant perturber le SPL ainsi que la performance globale de l'algorithme.

Nous pouvons par ailleurs visualiser des épisodes d'évaluation avec script fourni par le Habitat API (Savva et al. [15]). La figure 22 montre quelques épisodes :

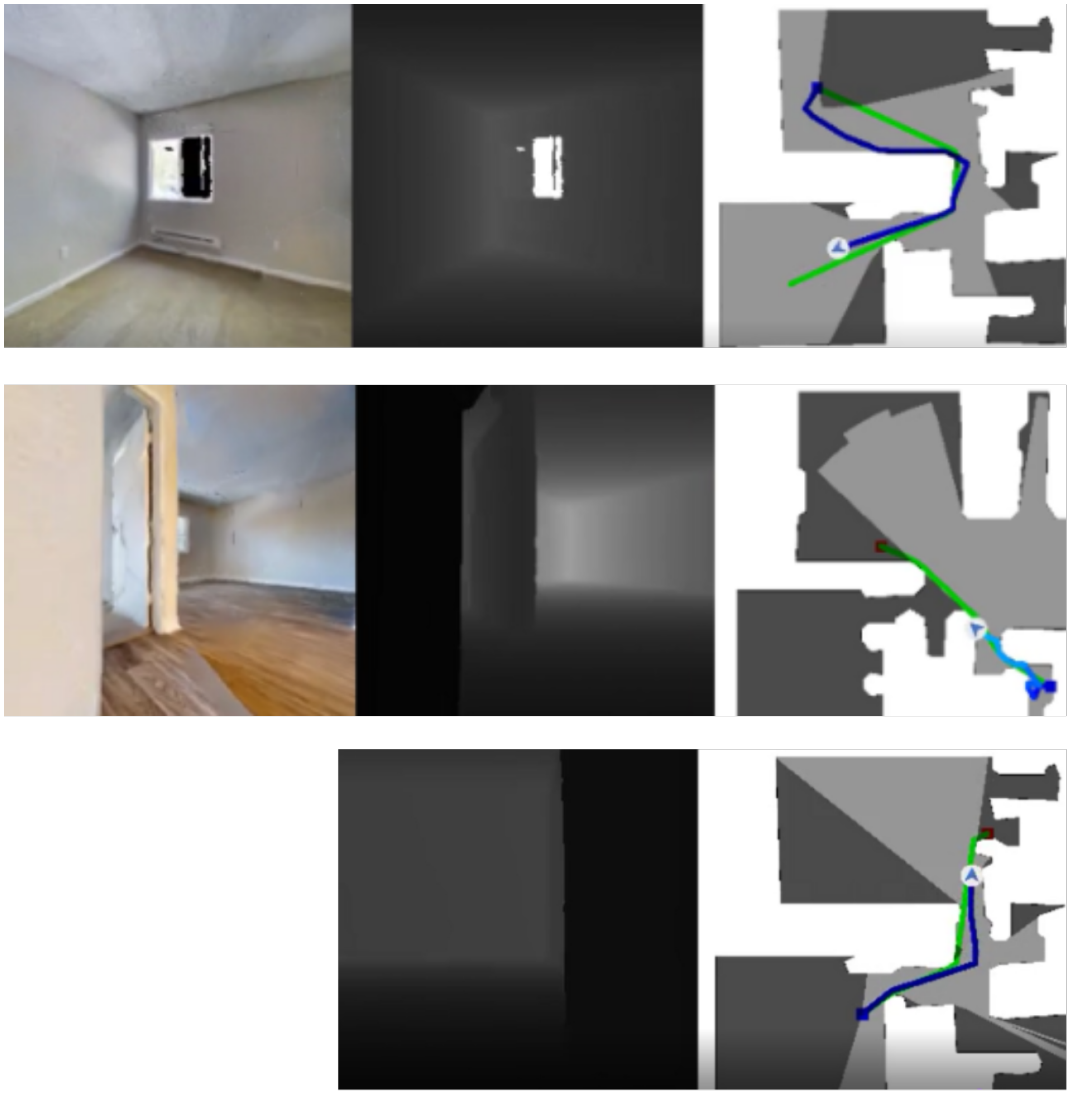


FIGURE 22: Exemples de scénarios réalisés dans Habitat, la trajectoire en vert est le plus court chemin, la trajectoire en bleue est celle choisie par l'agent

Sur les épisodes de la figure 22, nous avons remarqué un comportement qui peut être problématique. Lors d'un passage en angle droit (premier épisode de la figure 22), l'agent choisit une trajectoire rasante par rapport à cet obstacle et peut percuter l'angle formé par le mur.

Afin d'éviter ce comportement nous avons augmenté artificiellement dans le simulateur Habitat le diamètre de l'agent en lui donnant une dimension environ deux fois supérieure à la dimension du robot réel et cela uniquement pour la phase d'entraînement. Après cette modification, le robot en inférence (donc avec un diamètre plus faible que pendant l'entraînement) choisit des trajectoires plus larges vis-à-vis des obstacles (dernier épisode de la figure 22).

TABLE 2: Influence de la réduction de diamètre en inférence sur le critère de sécurité, l'évaluation est effectuée sur une même dataset de 1000 épisodes variés et avec la fonction de récompense SOA

Comparaison performance après diminution du diamètre du robot pour l'inférence				
Diamètre de l'agent	Récompense (moyenne)	SPL (moyen)	Taux de succès (moyen)	Nombre de collisions (moyen)
Diamètre inchangé 0.2 m	7.05	0.86	0.96	18.4
Diamètre réduit 0.1 m	7.2766	0.7957	0.88	16.2

Sur la table 2, nous pouvons voir que l'on peut aussi diminuer le nombre de collisions en réduisant le diamètre du robot uniquement pour l'inférence. Sur une dataset d'évaluation de 1000 épisodes, le robot a diminué de 20 % le nombre de collisions par rapport au robot dont le diamètre n'a pas été modifié entre la phase d'entraînement et la phase d'inférence. On remarque ques les autres indicateurs restent bons. Le robot choisit une trajectoire plus large, ce qui est le comportement recherché.

Nous pensons également pouvoir améliorer la sécurité du robot en tenant compte de la distance aux plus proches obstacles, en ajoutant le terme suivant dans la fonction de récompense $\mathbf{1}_{prev} w_{prev} r_{prev}$, nous pouvons l'appeler le terme préventif avec $r_{prev} = k \cdot d_{obstacles}$ et $\mathbf{1}_{prev}$ est la fonction indicatrice du seuil de danger qui vaut 1 si l'obstacle est assez proche et 0 sinon.

$$r(s,a) = w_{sp} r_{sp} + \mathbf{1}_{suc} r_{suc} + w_d r_d + \mathbf{1}_{col} r_{col} + \mathbf{1}_{prev} w_{prev} r_{prev}$$

Nous avons défini un capteur de plus courte distance aux obstacles basé sur la carte de profondeur (recherche de minimum dans la carte de profondeur). Nous aurions souhaité entraîner l'agent avec cette fonction, mais le capteur créé ralentit trop l'entraînement et empêche la convergence.

4.4.4 Inférence et intérêt de la multi-modalité : RGB + D

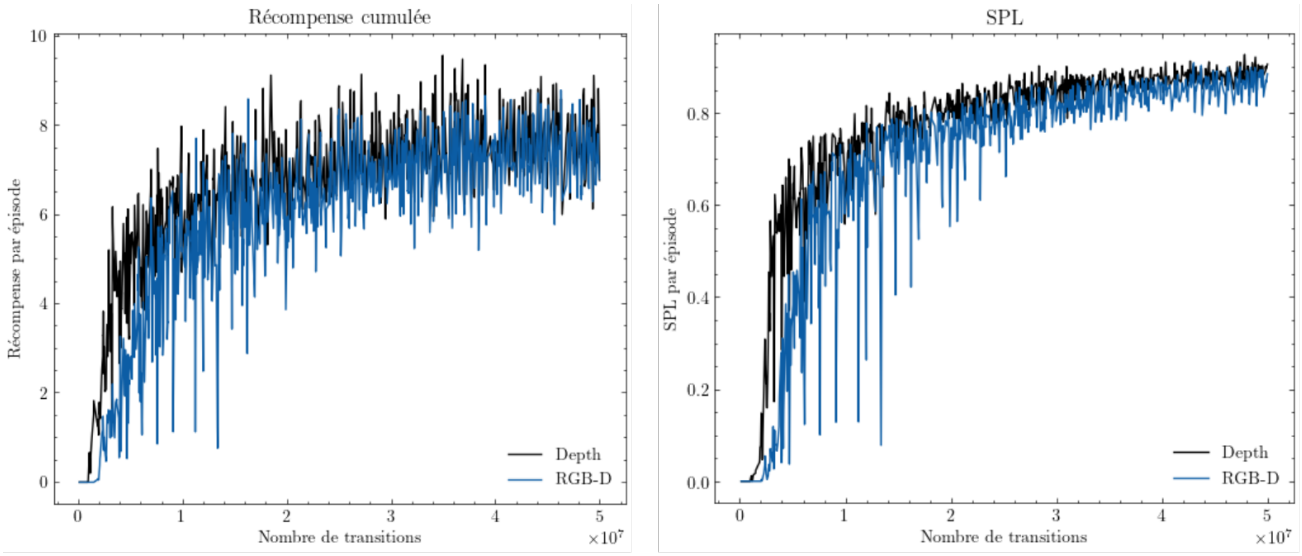


FIGURE 23: Comparaison avec la récompense sur le graphe de gauche et sur le graphe de droite la comparaison est effectuée avec le SPL

Nous pouvons voir, sur la figure 23, l'impact sur la performance qu'a la multi-modalité : une image RGB combinée à une carte de profondeur. Nous avons un ralentissement faible de l'entraînement en utilisant la profondeur en plus. Néanmoins, cet ajout impacte la vitesse de convergence et donne des performances équivalentes voire légèrement plus faibles. Cette multimodalité pourrait être utile dans le cas d'un capteur de profondeur peu fiable avec de nombreux pixels non-définis. Nous pourrions prolonger l'étude en testant l'image RGB seule et la combinaison avec un capteur sémantique (cf. Bianco et al. [1]). Wijmans et al. [20] suggèrent une chute importante de la performance en utilisant une image RGB seule par rapport à la profondeur lors de l'augmentation de la difficultés des épisodes. Cela justifie notre choix de baser notre entraînement uniquement sur un capteur de profondeur.

4.4.5 Inférence et intérêt de l'optimisation du *backbone*

TABLE 3: Importance du *backbone CNN* dans l'architecture du réseau de neurones

	Backbone ResNet18	Inférence totale
Temps inférence (ms)	9	11
Nb de paramètres (en M)	3.1	5.8

Nous pouvons voir sur la table 3, l'impact qu'a le *backbone* sur le coût calculatoire d'une inférence. Le temps d'inférence est une moyenne sur 1000 inférences (CPU Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz et 1 GPU GeForce RTX 3090 25 GB). Nous observons qu'une inférence sur le *backbone* ResNet18 occupe 86 % du temps de l'inférence totale. De plus, le *backbone* occupe 53 % de la quantité totale de paramètres du réseau de neurones. Le *backbone* est donc bien un

élément critique, nous devons trouver un équilibre entre la bonne extraction des features visuelles et la réduction du coût calculatoire.

4.4.6 Hyperparamètres de l'algorithme PPO

Afin de reproduire l'expérience, voici la liste des hyperparamètres utilisés (table 4), ce sont les hyperparamètres proposés par Wijmans et al. [20], ils sont semblables à ceux utilisés par Schulman et al. [16] alors que la tâche était Gym-Mujoco :

TABLE 4: Hyperparamètres

Paramètres	Valeur
optimiseur	Adam (Kingma and Ba [6])
learning rate	$2.5 \cdot 10^{-4}$
discount (γ)	0.99
paramètre GAE (λ)	0.95
paramètre de clipping (ϵ)	0.2
nombre de couches cachées (acteur-critique)	1
nombre de cellules par couche cachée	512
nombre de travailleurs	4
nombre total de transitions	$75 \cdot 10^6$
horizon T	128
non-linéarité	ReLU
pénalité de vitesse ($w_{sp} r_{sp}$)	-0.01
récompense du succès (r_{suc})	2.5
pénalité de collision (r_{col})	-1

5 Expérimentations et évaluations

5.1 Transfert dans Gazebo

Avant d'effectuer des inférences dans le monde réel, nous souhaitons valider les performances obtenues dans Habitat sur un simulateur intégrant un moteur physique, avec notre robot modélisé de manière réaliste et avec une reproduction la plus fidèle possible des épisodes que l'agent rencontrera dans le monde réel(figure 24).

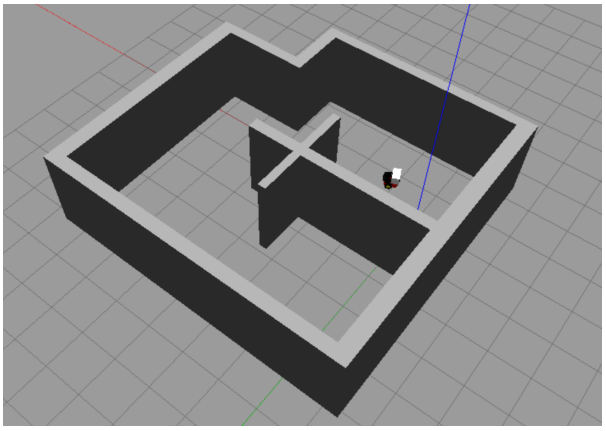


FIGURE 24: Reproduction réaliste du monde réel de l'environnement et de l'agent (dimensions, placement des capteurs) dans Gazebo

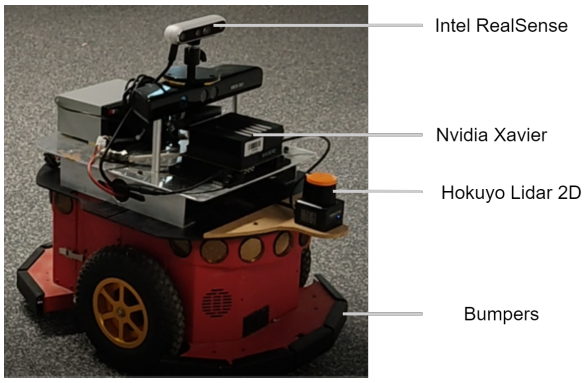


FIGURE 25: Description matérielle du robot

5.1.1 Description matérielle et modélisation de l'agent

L'agent est composé des éléments suivants (figure 25) :

- Plateforme Robotique : Pioneer 3-DX [4], dimensions 455 mm x 381 mm x 237 mm, poids-vide : 9kg, 2 roues et commande différentielle en vitesse, vitesse de rotation : 300 degrès par seconde, vitesse d'avance maximale 1.2 m/s, rayon de courbure lors des rotations : 0 m (il peut tourner sur lui même)
- Camera stéreo de la profondeur : Intel RealSense [5]
- LIDAR 2D : Hokuyo UST-10LX, spectre de 270 degrès pour une portée de 10m
- Ordinateur embarqué : NVIDIA Jetson Xavier AGX [6]
- OS : Ubuntu 18.04 équipé de ROS 1 (version Melodic)

5.1.2 Architecture logicielle

Nous construisons notre architecture logicielle avec le middleware ROS 1. Il présente l'avantage de facilement communiquer avec Gazebo et peu de modifications seront nécessaires pour le passage sur le robot réel.

4. https://www.generationrobots.com/media/Pioneer3DX-P3DX-RevA.pdf
5. https://www.intel.fr/content/www/fr/fr/architecture-and-technology/realsense-overview.html
6. https://www.nvidia.com/fr-fr/autonomous-machines/embedded-systems/jetson-agx-xavier/

Sur la figure 26, on a le robot et ses capteurs qui constituent des noeuds en interaction avec Gazebo. Nous avons un noeud nommé "Agent de navigation basé RL" qui s'abonne aux topics publiés par le noeud du capteur de profondeur et par le noeud "LaserScan Matcher". Il publie son action discrète sous forme de clé de dictionnaire. Un noeud est chargé de faire le contrôle proportionnel en vitesse pour effectuer les actions discrètes qu'on a définies dans Habitat.

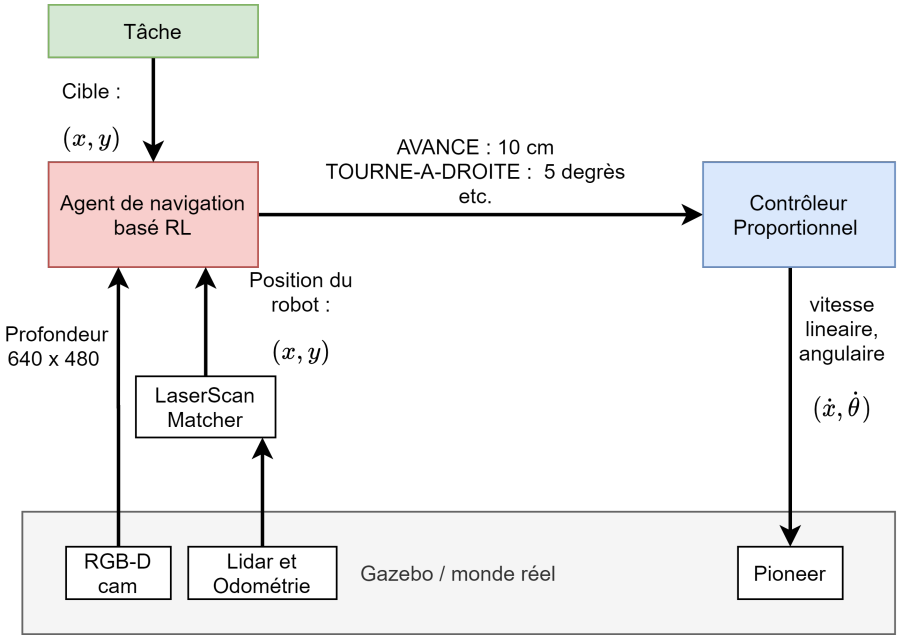


FIGURE 26: Architecture logicielle utilisée

5.1.3 Comportement de l'agent

L'agent a perdu son niveau de performance lors du passage du simulateur Habitat à Gazebo. Le contournement d'un obstacle court est bien effectué.

Toutefois, lors du contournement d'un obstacle long, le robot a tendance à effectuer des allers et retours le long de cet obstacle en cycle. Cela peut s'expliquer de la manière suivante : en longeant le mur pour suivre la trajectoire imposée par l'architecture de l'environnement, l'agent s'éloigne de la cible en terme de distance L_2, sa récompense est donc négative, l'agent cherche alors un autre chemin et ainsi de suite.

Pour résoudre ce problème, sans changer les poids calculés durant l'entraînement, nous pouvons augmenter la distance parcourue lors de chaque étape. Relativement à la longueur de l'obstacle la distance d'avance est plus faible. Cela revient à se placer dans le cas précédent d'un obstacle plus court et l'agent peut alors contourner cet obstacle.

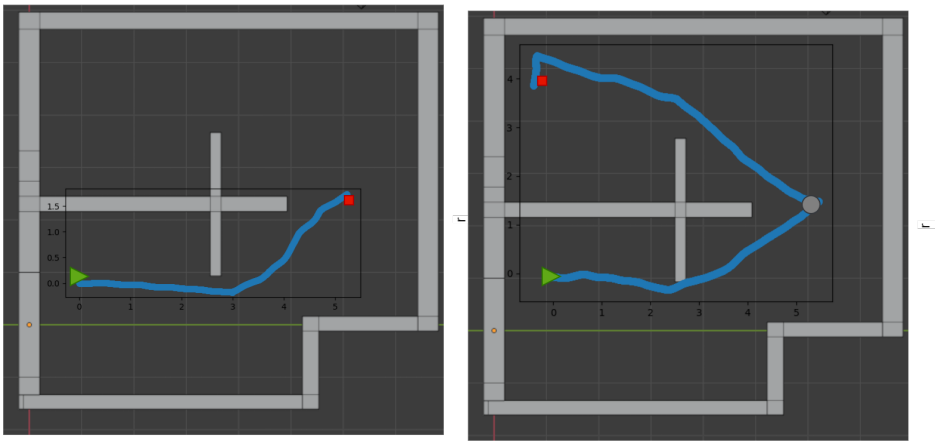


FIGURE 27: Contournement d'une cloison dans Gazebo. La ligne bleue est la trajectoire du robot. Le triangle vert est le point de départ, le carré rouge la cible et le cercle gris un point de passage.

Ces expériences dans Gazebo nous ont permis de valider le comportement que l'on a observé dans le simulateur Habitat ainsi que notre architecture logicielle ROS (figure 26). Le robot peut naviguer dans l'environnement sans percuter d'obstacles et rejoindre des cibles séparées par des cloisons (figure 27). Néanmoins nous n'avons pas pu dans le temps imparti construire un tableau de statistiques sur un nombre significatif d'inférences comme nous avons pu le faire dans Habitat.

5.2 Transfert dans le monde réel

En obtenant des trajectoires satisfaisante dans Gazebo, nous pouvons être confiants sur le fonctionnement du robot dans le monde réel.

5.2.1 Architecture du monde

Dans une salle d'expérience, nous avons reproduit l'environnement Gazebo. Les dimensions du monde sont d'environ 6 m x 7 m avec des cloisons amovibles afin de pouvoir construire plusieurs scénarios (figure 24). Les cloisons sont des fins panneaux de polystyrène, donc facilement déplaçables et non-impactants pour le robot en cas de chocs (figure 28).

FIGURE 28: Photo du monde réel, nous pouvons voir les cloisons en polystyrène et un *waypoint* sur le sol

5.2.2 Méthode utilisée

Avant de lancer l'algorithme de navigation, nous avons vérifié les différents modules du graphe des noeuds (figure 26).

- Fonctionnement du noeud *laser-scan matcher*[7] chargé de la fusion lidar - odométrie pour avoir une bonne localisation. Nous n'avons pas remarqué de dérive et la position est correcte. *Laser-scan matcher* définit l'origine du monde lors de son initialisation.
- Fonctionnement contrôleur proportionnel. Si le robot doit avancer de 1 m l'erreur est au maximum d'une dizaine de cm, c'est suffisant.
- Le temps d'inférence est de 25 ms sur le réseau de neurones (table 5), nous pouvons donc fixer la fréquence de contrôle dans ROS de la politique à 40 Hz maximum. Le temps d'inférence est le facteur limitant car les données de capteurs arrivent à une plus haute fréquence. L'asservissement proportionnel tourne à 15 Hz. Nous avons calculé le gain du contrôleur proportionnel pour que la vitesse maximale du robot soit atteinte car nous connaissons la distance d'avance.

TABLE 5: Temps d'inférence sur le GPU embarqué

	Backbone ResNet18	Inférence totale
Temps inférence (ms)	14	25

Nous n'utilisons aucune optimisation avec des outils Nvidia comme TensorRT et CuDNN mais le portage simple de PyTorch. Avec ce travail supplémentaire, nous pourrions réduire le temps d'inférence.

7. http://wiki.ros.org/laser_scan_matcher

Dès lors, nous pouvons commencer les expériences de navigation.

5.2.3 Scénarios mis-en-place et résultats de navigation

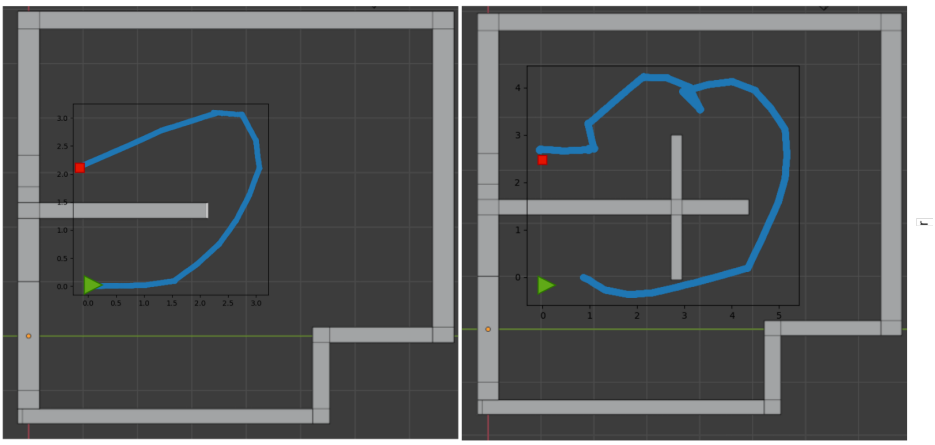


FIGURE 29: Contournement d'une cloison dans le monde réel. La ligne bleue est la trajectoire du robot, le triangle vert est le point de départ et le carré rouge la cible.

Nous avons mis-en-place plusieurs scénarios de test. Les premiers tests ont été effectués sans obstacle. Sur ce scénario, le robot navigue de manière optimale en prenant le plus court chemin. Nous avons également essayé de reproduire les scénarios complexes validés dans Gazebo en ajoutant des cloisons. Sur la figure 29 de gauche, nous observons que le contournement d'une cloison de 2.5 m est bien effectué la trajectoire est directe. Sur la figure 29 de droite, le contournement de la cloison avec la croix est effectué avec plus de difficultés. En effet, la forme de la cloison, laisse un faible passage pour le robot, ce passage apparaît difficilement dans la carte de profondeur. Ce type de scénario montre les limites de l'agent sans l'aide d'un planificateur global. Un *waypoint* intermédiaire assurerait une meilleure navigation. Néanmoins, nous avons validé qu'en supprimant la cloison, le robot choisit la trajectoire la plus directe alors qu'avec une approche modulaire une nouvelle planification est nécessaire.

6 Conclusion et perspectives

6.1 Conclusion

Dans ce stage, nous nous sommes concentrés sur l'application de l'apprentissage par renforcement à la conduite d'un robot mobile dans un environnement intérieur.

La plupart des travaux existants dans la littérature sur la navigation autonome de bout en bout utilisent l'apprentissage par renforcement avec un module de perception augmenté par rapport à une image RGB seule. Ils utilisent par exemple de la sémantique ou une carte de profondeur. Les algorithmes choisis sont des acteurs-critiques et l'espace d'action est discrétisé.

Plusieurs aspects de la navigation autonome ont été étudiés : la qualité de la trajectoire, la capacité de généralisation à un environnement inconnu, la capacité du robot à naviguer en sécurité.

Nous avons choisi la modalité de notre agent et les actions que nous voulions. Afin d'avoir une trajectoire plus fluide, nous avons commencé par travailler sur un espace d'action continu. Cela impose un algorithme à politique stochastique.

En ce sens, nous avons donc initialement développé notre propre implémentation de l'algorithme soft-actor-critic (d'après Haarnoja et al. [4]) qui est un acteur-critique *off-policy* (la mise-à-jour de la politique ne dépend pas forcément des transitions définies par la politique la plus récente) avec une politique stochastique. Nous avons ensuite mené une étude d'hyperparamètres sur cet algorithme mais cela nécessitait trop de temps pour converger sur notre tâche complexe.

De ce fait, nous avons ensuite fait le choix d'utiliser un framework d'apprentissage par renforcement et d'utiliser l'algorithme *on-policy* PPO. Le choix d'un simulateur cinématique performant nous a permis de reproduire les résultats de l'état de l'art. En définissant notre fonction de récompense avec un terme de collision, nous avons pu dépasser les résultats de l'état de l'art selon notre critère de sureté en réduisant drastiquement le nombre de collisions.

Enfin nous avons proposé un transfert progressif de l'environnement cinématique vers le monde réel en utilisant un simulateur dynamique intermédiaire pour valider méthodiquement les comportements du robot observés dans le simulateur cinématique, puis nous avons utilisé la méthode de transfert pour le passage du simulateur dynamique au monde réel.

6.2 Perspectives

Les travaux présentés pourraient être améliorés avec les points suivants. Si nous avions eu plus de temps, nous aurions souhaité les étudier.

Comparaison avec une navigation modulaire Nous pourrions comparer notre approche avec l'approche modulaire par exemple celle utilisée dans la couche de navigation de ROS (Zheng [23]). Les critères de comparaison seraient : la consommation d'énergie, la distance parcourue, le temps de trajet.

Fine-Tuning Afin d'améliorer le transfert sur le simulateur physique et dans le monde réel, nous pourrions mettre en place une méthode de *fine-tuning*, d'abord dans le simulateur physique (Gazebo) puis dans le monde réel. Il s'agit d'une technique très populaire en apprentissage supervisé améliorant les performances.

Améliorer la fonction de récompense Afin d'avoir une navigation plus certaine, nous pourrions tenir compte de la distance aux obstacles les plus proches dans la fonction de récompense. Dans le simulateur physique (Gazebo) et dans le monde réel, le LIDAR ou la stéréo camera peuvent nous donner cette distance aux obstacles mais le simulateur cinematique (Habitat) utilisé n'a pas de capteur performant donnant cette grandeur. Nous pourrions en développer un avec Habitat-API, ou bien, cette récompense améliorée ne serait utilisée que dans le *fine-tuning* de Gazebo.

Fusion entre un planificateur global et notre approche Nous avons remarqué que dans un environnement complexe (i.e. une distance L_2 à la cible courte mais une distance géodésique (A*) importante), notre agent perd de son intérêt car il effectue une exploration pour trouver une issue à la manière d'un agent construisant une carte. Notre agent est performant dans l'évitement d'obstacles dynamiques donc à l'échelle du planificateur local. Il pourrait être intéressant d'utiliser un planificateur local plaçant intelligemment (espacement adapté à la performance de l'agent) des *waypoints* intermédiaires qui sont les cibles de notre agent. Si la carte n'est pas disponible l'agent peut la construire pendant son exploration avec, par exemple, une technique de SLAM.

Actions choisies dans un espace continu La trajectoire du robot n'est pas fluide du fait de l'espace d'actions discrétisé. Nous pourrions facilement adapter notre algorithme (la politique est stochastique) pour choisir des actions dans un intervalle. Ces actions continues seraient une vitesse linéaire et une vitesse angulaire. Le contrôleur proportionnel que nous avons construit serait alors implicitement contenu dans le réseau de neurones. Cependant, cette méthode pourrait nécessiter un *fine-tuning* lors du transfert du simulateur cinématique au simulateur dynamique.

S'affranchir du capteur LIDAR Le LIDAR sur notre robot ne sert dans notre approche qu'à la localisation de l'agent. Cependant un capteur LIDAR est onéreux et gourmand en énergie, nous souhaiterions donc l'enlever du robot et localiser le robot autrement. Nous pourrions par exemple robustifier la localisation odométrique avec une fusion de capteurs : imu, odométrie visuelle. Nous pourrions également penser à un système de localisation du robot avec des AruCos au-dessus de la scène.

Références

[1] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6 :64270–64277, 2018. doi : 10.1109/ACCESS.2018.2877890.

[2] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL http://arxiv.org/abs/1412.3555.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[4] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic : Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL http://arxiv.org/abs/1801.01290.

[5] Catherine F. Higham and Desmond J. Higham. Deep Learning : An Introduction for Applied Mathematicians. *arXiv :1801.05894 [cs, math, stat]*, January 2018. URL http://arxiv.org/abs/1801.05894. arXiv : 1801.05894.

[6] Diederik P. Kingma and Jimmy Ba. Adam : A method for stochastic optimization, 2017.

[7] Jonáš Kulhánek, Erik Derner, and Robert Babuška. Visual navigation in real-world indoor environments using end-to-end deep reinforcement learning. *IEEE Robotics and Automation Letters*, 6(3) :4345–4352, 2021. doi : 10.1109/LRA.2021.3068106.

[8] Enrico Marchesini and Alessandro Farinelli. Discrete deep reinforcement learning for mapless navigation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10688–10694, 2020. doi : 10.1109/ICRA40945.2020.9196739.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

[10] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 1928–1937. JMLR.org, 2016.

[11] Urs Muller, Jan Ben, Eric Cosatto, Beat Flepp, and Yann Cun. Off-road obstacle avoidance through end-to-end learning. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems*, volume 18. MIT Press, 2005.

[12] Matthias Müller, Alexey Dosovitskiy, Bernard Ghanem, and Vladlen Koltun. Driving policy transfer via modularity and abstraction. 2018.

[13] Viktor Rausch, Andreas Hansen, Eugen Solowjow, Chang Liu, Edwin Kreuzer, and J. Karl Hedrick. Learning a deep neural net policy for end-to-end control of autonomous vehicles. In *2017 American Control Conference (ACC)*, pages 4914–4919, 2017. doi : 10.23919/ACC.2017.7963716.

[14] N. Sariff and N. Buniyamin. An overview of autonomous mobile robot path planning algorithms. In *2006 4th Student Conference on Research and Development*, pages 183–188, 2006. doi : 10.1109/SCORED.2006.4339335.

[15] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat : A platform for embodied ai research, 2019.

[16] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[17] Ralf C. Staudemeyer and Eric Rothstein Morris. Understanding LSTM - a tutorial into long short-term memory recurrent neural networks. *CoRR*, abs/1909.09586, 2019. URL http://arxiv.org/abs/1909.09586.

[18] Hartmut Surmann, Christian Jestel, Robin Marchel, Franziska Musberg, Houssem Elhadj, and Mahbube Ardani. Deep reinforcement learning for real autonomous mobile robot navigation in indoor environments. *CoRR*, abs/2005.13857, 2020. URL https://arxiv.org/abs/2005.13857.

[19] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning : An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

[20] Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. DD-PPO : Learning Near-Perfect PointGoal Navigators from 2.5 Billion Frames. *arXiv :1911.00357 [cs]*, January 2020. URL http://arxiv.org/abs/1911.00357. arXiv : 1911.00357.

[21] Fei Xia, Amir R. Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env : Real-world perception for embodied agents. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9068–9079, 2018. doi : 10.1109/CVPR.2018.00945.

[22] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. *CoRR*, abs/1612.01079, 2016. URL http://arxiv.org/abs/1612.01079.

[23] Kaiyu Zheng. ROS navigation tuning guide. *CoRR*, abs/1706.09068, 2017. URL http://arxiv.org/abs/1706.09068.

[24] Brady Zhou, Philipp Krähenbühl, and Vladlen Koltun. Does computer vision matter for action ? *Science Robotics*, 4(30), 2019. doi : 10.1126/scirobotics.aaw6661. URL https://robotics.sciencemag.org/content/4/30/eaaw6661.