Rapport de projet de fin d'étude

Année académique 2025/2026

NAVIGATION ROBOTIQUE TERRESTRE EN MILIEU DESTRUCTURÉ PAR APPRENTISSAGE PAR RENFORCEMENT

Confidentialité : Non Protégé

Léon PERRUCHOT-TRIBOULET

Promotion FISE 26

Spécialité Robotique

Période de stage	23 Mars 2026 – 28 Août 2026
Lieu d'affectation	Agence Ministérielle pour l'Intelligence Artificielle de Défense (AMIAD), Pôle Technique (Bruz)
Encadrant	Dr. Jorand GALLOU, Cellule Agents Autonomes et Aides à la Décision
Référent académique	Pr. Luc Jaulin, ENSTA – Lab-STICC
Jury	Président – Pr. Laurant HARDOUIN Rapporteur – Pr. Luc Jaulin

Résumé

Résumé

Ce travail, inscrit dans le cadre du projet Pendragon de l'Agence Ministérielle pour l'Intelligence Artificielle de Défense (AMIAD), vise à doter un drone terrestre à direction par dérapage (skid-steer) d'un contrôleur robuste pour la navigation autonome en milieu destructuré. L'étude s'articule en quatre axes complémentaires. D'abord, nous comparons plusieurs modélisations cinématiques et dynamiques proposées dans la littérature pour représenter le glissement inhérent à ce type de plateforme, et mettons en évidence leurs limites de généralisation face à la diversité des terrains rencontrés. Les contrôleurs classiques de la bibliothèque Nav2 (Dynamic Window Approach B et Model Predictive Control) sont ensuite analysés théoriquement puis déployés sur un robot Agilex Scout2.0, révélant leur sensibilité au réglage empirique et leur dépendance à une chaîne de localisation fiable. Puis un jumeau numérique du robot Clearpath Warthog est développé sous NVIDIA Isaac Sim puis Isaac Lab, incluant l'automatisation de la conversion URDF vers USD, la génération procédurale de terrains segmentés par diagramme de Voronoï et un capteur de carte de coût sémantique. Cet environnement sert enfin à l'entraînement d'agents par apprentissage par renforcement (algorithme PPO), dont la politique de commande est affinée par curriculum progressif sur la distance à l'objectif et par une conception itérative de la fonction de récompense. Les résultats obtenus ne dégagent pas de méthode universellement supérieure, mais révèlent un compromis entre effort de réglage paramétrique, propre aux approches classiques, et effort de conception architecturale, caractéristique de l'apprentissage par renforcement.

Mots-clefs : robotique mobile, direction par dérapage, apprentissage par renforcement, simulation, Isaac Sim, Nav2

Abstract

This work is part of the Pendragon project led by the French Ministerial Agency for Military Artificial Intelligence (AMIAD), aiming to equip a skid-steer unmanned ground vehicle with a robust controller for autonomous navigation in unstructured environments. The study is structured around four complementary axes. A first modeling phase compares several kinematic and dynamic formulations from the literature intended to capture the wheel slippage inherent to skid-steer platforms, highlighting their limited generalization across varied terrain conditions. Classical Nav2 controllers (Dynamic Window Approach B and Model Predictive Control) are then analyzed theoretically and deployed on an Agilex Scout2.0 platform, exposing their sensitivity to empirical tuning and their dependence on reliable localization. A digital twin of the Clearpath Warthog robot is subsequently built in NVIDIA Isaac Sim and Isaac Lab, including automated URDF to USD conversion, procedural generation of terrains segmented via Voronoi diagrams, and a custom semantic costmap sensor. This environment is finally used to train reinforcement learning agents with the PPO algorithm, whose control policy is refined through a progressive curriculum on goal distance and iterative reward shaping. The results do not identify a universally superior method, but instead reveal a trade-off between the parametric tuning effort characteristic of classical approaches and the architectural design effort inherent to reinforcement learning.

Keywords : mobile robotics, skid-steer navigation, reinforcement learning, simulation, Isaac Sim, Nav2

Remerciements

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

Je suis avant tout reconnaissant envers mon tuteur Jorand GALLOU pour son accompagnement, sa confiance et sa disponibilité tout au long de mon stage. Ses conseils m'ont permis d'approfondir mes connaissances et de découvrir de nombreux sujets aussi enrichissants que passionnants.

Je tiens également à exprimer ma gratitude envers toute l'équipe de la cellule AAAD au sein de laquelle j'ai été intégré, pour leur accueil chaleureux, leur bonne humeur et pour m'avoir offert l'opportunité de découvrir leurs projets.

Merci aux responsables de l'AMIAD qui m'ont donné l'opportunité de participer à ICRA 2026.

J'adresse enfin mes remerciements aux différentes personnes rencontrées durant ce stage, ingénieurs comme stagiaires, pour la richesse de nos échanges, qui ont rendu cette expérience aussi agréable que motivante et m'ont fait comprendre l'étendue d'un programme robotique.

Notice sur l'intelligence artificielle générative

Des outils d'intelligence artificielle générative ont été utilisés dans le cadre de ce travail.Leur usage a été limité à l'optimisation de programmes informatiques, à la recherche accélérée au sein de documentations techniques, ainsi qu'à la correction syntaxique et orthographique du présent rapport.

Table des matières

Chapitre 1

Introduction

1.1 Parcours et motivations

La spécialisation robotique de la formation d'ingénieur généraliste de l'ENSTA Bretagne est fondamentalement pluridisciplinaire et couvre un large éventail de domaines d'application de la robotique mobile autonome. Elle propose un équilibre idéal entre enseignements théoriques et pratiques, combinant cours magistraux, travaux pratiques et projets d'application variés. La spécialité étant axée sur la robotique marine, l'étude de robots mobile terrestres n'a que très peu été abordé en cours. Motivé par ma curiosité de découvrir de nouvelles modalités robotiques et des usages de l'Intelligence Artificielle (IA) dans ce domaine, j'ai ainsi choisi de réaliser mon stage de fin d'études au sein de l'Agence Ministérielle pour l'Intelligence Artificielle de Défense (AMIAD).

1.2 Présentation de l'entité d'accueil

L'Agence Ministérielle pour l'Intelligence Artificielle de Défense (AMIAD) est l'autorité technique de référence sur la question de l'IA au sein du Ministère des Armées. L'agence est responsable de la stratégie d'IA du ministère, une charge qui regroupe à la fois les questions d'usage, de sécurité, mais aussi d'éthique relative à son utilisation. Elle possède aussi les moyens de production et d'achat de solutions intégrants des produits d'IA au profit de toutes les armées. Disposant de l'ensemble des compétences allant de l'élaboration et l'exploitation de super calculateurs, au déploiement d'outils sur des plateformes embarquées, en passant par l'entraînement de modèles souverains, l'AMIAD est le chef-lieu du développement de ces technologie pour le ministère.

De plus l'AMIAD est répartie sur deux sites, un premier sur le plateau de Saclay où se trouve le pôle de recherche avec notamment un laboratoire conjoint avec l'École Nationale Supérieure de Techniques Avancées (ENSTA), le LARIAD. Et un second à Bruz sur un site de la DGA responsable des projets techniques. Sa proximité directe avec, à la fois les activités de recherche du plateau de Saclay, et les activités industrielles du site de DGA-MI lui permet d'être au coeur des enjeu de l'IA de défense.

1.3 Contexte

Parmi la multitude de projets traités par l'AMIAD, le projet *Pendragon* cherche à établir la première Unité Robotique de Combat (URC). C'est à dire, un ensemble de plateformes robotiques terrestre et aériens capables de manière autonome et collaborative de remplir diverses missions, allant de la reconnaissance d'environnements, à l'appui tactique.

Pour mener à bien un projet d'un telle ampleur il est nécessaire d'assembler progressivement des briques d'autonomie. C'est pourquoi ce stage se limite à l'étude d'un type particulier de Drone Terrestre (DT), ceux ayant une direction par dérapage (ou *skid-steer*).

La robotique mobile terrestre est un domaine de la robotique très étudié avec de nombreuses applications industrielles matures. On retrouve ainsi de nombreux robots dans des entrepôts pour de l'aide logistique, dans des domiciles pour aider au ménage ou dans des hôpitaux pour assister des patients ou du personnel médical. Au contraire de ces milieux confinés et structurés la robotique mobile autonome en plein air doit faire face à de nombreuses contraintes supplémentaires. Les robots évoluent dans des environnements potentiellement infinis et plus variés. De plus le manque de contrôle de l'environnement demande une plus grande robustesse des différents systèmes du robot (perception, contrôle, localisation...) pour faire face aux imprévus.

1.4 Objectifs

C'est dans ce contexte que l'on cherche à mettre en place un contrôleur robuste pour permettre à un Unmanned Ground Vehicle (UGV) de naviguer en milieu destructuré de manière autonome. Pour ce faire nous essayerons de répondre aux objectifs suivants :

— Modéliser la plateforme robotique
— Simuler le robot et l'ensemble de ses capteurs
— Analyser des contrôleurs classiques
— Créer un modèle de Machine Learning (ML) pour contrôler le robot
— Comparer les contrôleurs mis en jeu

Ce stage de vingt-trois semaines s'est déroulé en plusieurs phases. Une représentation graphique de la planification prévue et effective du stage est représentée en annexe dans le diagramme .1.

Chapitre 2

Méthodes classiques

2.1 Modélisation cinématique et dynamique

Pour ce travail nous avons étudié deux robots, le Clearpath Warthog[1] et l'Agilex Scout V2[2]. Ces deux robots ont pour particularité d'être à direction par dérapage (skid steer). Leur capacité à se déplacer et à tourner repose sur la variation des vitesses de rotation des roues situées de part et d'autre du châssis.

Un robot à direction par dérapage ne peut cependant pas être directement assimilé à un robot différentiel tel le Turtlebot[3]. Ce dernier peut être modélisé simplement par les équations 2.1 dérivées de [4], sous l'hypothèse d'un roulement sans glissement.

$$\begin{cases} \dot{x} = \dfrac{v_l + v_r}{2}\cos(\theta), \\ \dot{y} = \dfrac{v_l + v_r}{2}\sin(\theta), \\ \dot{\theta} = \dfrac{v_r - v_l}{2b}. \end{cases} \tag{2.1}$$

Avec l'état du robot donné par $\mathbf{x} = [x, y, \theta]^T$, où (x, y) sont les coordonnées cartésiennes du centre du robot, θ son cap dans le repère monde, e la demi-distance entre les roues, et (v_l, v_r) les vitesses d'avance respectives des roues gauche et droite.

Cette modélisation permet, à partir des vitesses des roues et de la géométrie du robot, de déterminer directement son mouvement dans le plan, sous l'hypothèse de roulement sans glissement. Cette hypothèse est toutefois difficilement applicable à un robot à direction par dérapage. En effet, lors d'une rotation, la géométrie et le positionnement des roues ne permettent pas de satisfaire simultanément les contraintes de roulement sans glissement pour l'ensemble des roues autour d'un même Centre Instantané de Rotation (CIR). Des phénomènes de glissement, apparaissent alors au niveau des roues.

Ce glissement ne peut par ailleurs pas être considéré comme une quantité constante, son comportement dépend notamment des caractéristiques du sol et des conditions de déplacement du robot. Il devient ainsi difficile d'établir une relation directe entre les vitesses de rotation des roues et la vitesse de déplacement du châssis à partir de la seule géométrie du robot.

Dans ce contexte, plusieurs modélisations simplifiés ont été proposés afin de prendre en compte, de manière plus ou moins explicite, les effets du glissement. Dans les paragraphes suivants, nous présentons trois de ces approches et discutons leurs hypothèses ainsi que leurs limites. Le schéma 2.1 représente les principaux paramètres du robot.

Approche cinématique comparative multi-modèles

Baril *et al.* [5] proposent une étude comparative de quatre modèles cinématiques appliqués à un robot à direction par dérapage lourd (590 kg), le *Warthog*, avec pour objectif d'évaluer leur validité sur des terrains aux propriétés différentes (neige et béton). Il est proposé de généraliser le modèle différentiel idéal de

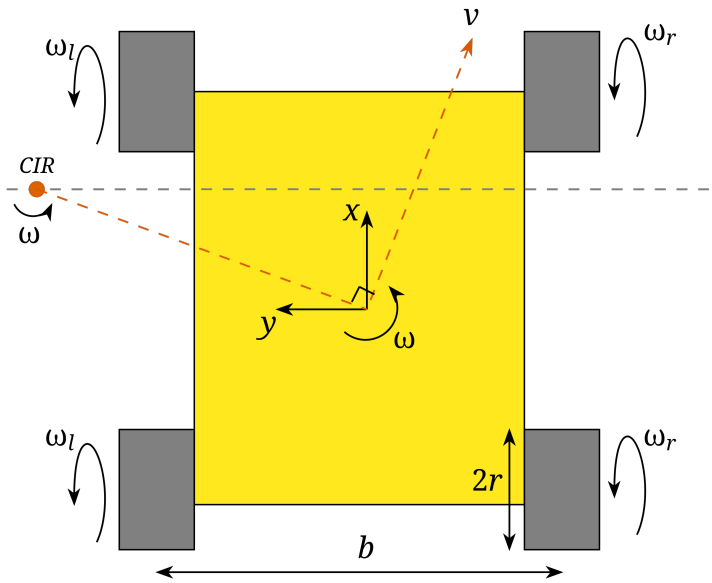


FIGURE 2.1 – Schéma des principaux paramètres de modélisation d'un robot à direction par dérapage

l'équation 2.1, qui suppose un roulement sans glissement, en relâchant l'hypothèse d'un CIR unique et confondu avec le centre géométrique du robot. Chaque train de roues se voit ainsi associer son propre CIR, supposé situé sur une droite parallèle à l'axe y local et invariant pour un terrain donné.

Ce modèle dit *extended differential-drive* s'écrit alors, dans sa version symétrique à deux paramètres :

$$\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = r\alpha \begin{bmatrix} 1/2 & 1/2 \\ 0 & 0 \\ -1/\hat{b} & 1/\hat{b} \end{bmatrix} \begin{bmatrix} \omega_l \\ \omega_r \end{bmatrix}, \tag{2.2}$$

où $\alpha \in [0, 1]$ est un paramètre de glissement, et $\hat{b}$ désigne la largeur effective du véhicule, distincte de sa largeur géométrique réelle. Ces deux paramètres sont identifiés hors-ligne par minimisation de la distance de Mahalanobis entre la trajectoire prédite et la trajectoire de référence obtenue par recalage Iterative Closest Point (ICP).

Une extension du modèle, dite *ROC-based*, fait dépendre la position du CIR du rayon de courbure instantané de la trajectoire λ

$$\begin{aligned} \lambda &= \left| \frac{\omega_r + \omega_l}{\omega_r - \omega_l} \right|, \\ \hat{b} &= b \left(1 + \frac{\beta_1}{1 + \beta_2 \sqrt{\lambda}} \right). \end{aligned} \tag{2.3}$$

et permet ainsi une adaptation continue du modèle en fonction du régime de manoeuvre.

Enfin, un modèle linéaire complet entièrement paramétrique est proposé :

$$\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = \begin{bmatrix} \gamma_{11} & \gamma_{12} \\ \gamma_{21} & \gamma_{22} \\ \gamma_{31} & \gamma_{32} \end{bmatrix} \begin{bmatrix} \omega_l \\ \omega_r \end{bmatrix}. \tag{2.4}$$

La principale motivation de ces approches est de rester purement cinématique, donc peu coûteuse en calcul. Elles ne requièrent ni connaissance des forces de contact ni identification de paramètres dynamiques, tout en intégrant les effets du glissement par les différents coefficients.

Les limites identifiées par les auteurs sont cependant notables. À l'exception du modèle asymétrique, et du modèle linéaire complet, ces formulations imposent une vitesse latérale nulle ($v_y = 0$), ce qui ne permet pas de représenter le glissement latéral. Par ailleurs, l'identification des paramètres s'avère fortement dépendante du choix des horizons temporels et spatiaux retenus pour la segmentation des trajectoires d'entraînement.

De plus cette étude propose des modèles pour deux natures de terrain différentes (béton et neige) et nécessiterait alors autant de modèles que de variation de terrains. L'auteur de ce papier a d'ailleurs étendu son étude en 2024 pour couvrir d'autres types de terrain avec ce papier [6]. Toutefois dans le cadre de la navigation destructurée étudiée lors de ce stage, le robot est non seulement amené à évoluer sur une plus grande variété de types de sols (gravier, herbe, forêts, asphalte...), mais il n'est aussi pas garanti que le robot arrive à déterminer celui-ci afin de changer de contrôleur en conséquence.

Modèle dynamique non linéaire

Arbinolo [7] développe une modélisation complète, structurée en un sous-modèle cinématique et un sous-modèle dynamique, respectivement issus de la géométrie du châssis et du formalisme d'Euler-Lagrange. Le vecteur d'état généralisé $q = [x, y, \theta]^T$, exprimé dans le repère global, est relié au vecteur de vitesse locale $v = [v_x, v_y, \omega_z]^T$ par l'intermédiaire de la position du CIR, dont l'existence impose une contrainte non-holonome sur le mouvement du système :

$$v_y + x_{CIR}\,\omega_z = 0. \tag{2.5}$$

Cette contrainte permet de définir un vecteur de commande cinématique réduit $\eta = [v_x, \omega_z]^T$, analogue à celui du modèle différentiel classique :

$$\eta = \begin{bmatrix} v_x \\ \omega_z \end{bmatrix} = r \begin{bmatrix} \dfrac{\omega_l + \omega_r}{2} \\ \dfrac{\omega_r - \omega_l}{2c} \end{bmatrix}. \tag{2.6}$$

La composante dynamique du modèle est obtenue à partir de l'équation d'Euler-Lagrange $\Gamma = \frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q}$, le Lagrangien L se réduisant à l'énergie cinétique du système sous l'hypothèse de mouvement planaire. Il est alors possible d'établir La formulation dynamique suivante, exprimée en coordonnées locales :

$$\mathbf{M}\dot{\eta} + \mathbf{C}\eta + \mathbf{R} = \mathbf{E}\tau, \tag{2.7}$$

où $\mathbf{M}$ désigne la matrice d'inertie généralisée, $\mathbf{C}$ regroupe les termes de Coriolis et centrifuges, $\mathbf{R}$ le vecteur des forces résistives, comprenant la résistance au roulement longitudinal et la force de friction latérale, et $\mathbf{E}$ la matrice de transformation reliant les couples moteurs $\tau = [\tau_L, \tau_R]^T$ aux forces généralisées appliquées au châssis. La motivation de cette approche est double. D'une part, elle vise à produire un modèle physiquement interprétable et directement exploitable pour la synthèse de lois de commande, notamment par linéarisation autour d'un point de fonctionnement et représentation d'état ; ce qui est utilisé à l'intérieur des contrôleurs propriétaires des robots. D'autre part, elle introduit une formulation de frottement sec améliorée, capable de distinguer les régimes d'adhérence et de glissement dynamique. Cette approche repose toutefois sur un ensemble d'hypothèses simplificatrices qui en délimitent le domaine de validité. Ainsi l'auteur reconnaît la symétrie longitudinale et latérale du châssis et le CIR supposé fixe et confondu avec le centre de gravité du robot (imposant $v_y = 0$).

Ainsi la complexité de ce modèle permet certes de décrire plus finement certains phénomènes de glissement mais ne permet pas de construire un contrôleur adapté à tous les terrains. Il faut cependant relever que c'est sûrement une modélisation semblable qui est utilisé à l'intérieur du robot pour établir les couples nécessaires à envoyer à chaque moteur à une commande donnée et pour l'odométrie proposée par le constructeur.

Modèle dynamique du pneu

Prado *et al.* [8] proposent une modélisation à l'échelle de chaque pneu, destinée à une commande prédictive non linéaire sur sol meuble. Le modèle du châssis reprend une structure de type unicyle étendue, avec

des incertitudes ajoutées δ représentant les effets terra-mécaniques non modélisés :

$$\dot{z}(t) = f(q(t), u(t), \Theta) + \delta, \qquad \xi(t) = g(q(t), u(t), \Theta) + \upsilon, \tag{2.8}$$

le vecteur d'état q étant décomposé en trois sous-systèmes couplés : le glissement longitudinal s_{ij} de chaque pneu, l'angle de dérive α_{ij}, et la pose et vitesse du châssis $[x, y, \theta, v_x, v_y, \omega_z]^T$. Les forces de traction et de dérive de chaque pneu sont modélisées à l'aide d'une version simplifiée du modèle de pneu.

$$\begin{aligned} F_{ijx} &= \mu_x(\sigma_{sij}) N_{ijz}, & \mu_x(\sigma_{sij}) &= D_x \sin\big(C_x \arctan(B_x \sigma_{sij})\big), \\ F_{ijy} &= \mu_y(\alpha_{ij}) N_{ijz}, & \mu_y(\alpha_{ij}) &= D_y \sin\big(C_y \arctan(B_y \alpha_{ij})\big). \end{aligned} \tag{2.9}$$

où le taux de glissement σ_{sij} et l'angle de dérive α_{ij} sont calculés à partir de l'écart entre la vitesse de roulement linéarisée $\omega_{ij} r$ et la vitesse linéaire réelle du pneu. La force normale N_{ijz}, quant à elle, n'est pas supposée constante mais dérivée du modèle de contact de Hunt-Crossley, qui capture l'enfoncement élastique du pneu dans un sol déformable :

$$N_{ijz} = -K_N \delta_{ij}^n - D_N \delta_{ij}^p \dot{\delta}_{ij}^q, \tag{2.10}$$

avec K_N et D_N la raideur et l'amortissement du sol, et δ_{ij} la profondeur de pénétration du pneu.

La motivation de cette approche est de disposer d'un modèle suffisamment riche pour représenter explicitement le couplage glissement/dérive/déformation du sol, nécessaire pour une commande robuste capable de compenser activement les pertes de traction en trajectoire. Les limites de cette approche résident dans sa complexité : un grand nombre de paramètres à identifier, et un coût de calcul élevé.

2.2 Contrôleurs classiques

À partir de la modélisation cinématique ou dynamique du robot il devient possible d'établir des contrôleurs pour diriger le robot. Cette section propose l'étude de deux approches classiques de contrôle local pour la navigation autonome des robots de type direction par dérapage : l'algorithme Dynamic Window approach B (DWB) et le Model Predictive Control (MPC). Ces deux méthodes partagent le même objectif, générer à chaque instant une consigne de vitesse admissible permettant de suivre une trajectoire ou d'atteindre un objectif tout en évitant les obstacles. Elles reposent tout de même sur des principes algorithmiques différents, respectivement réactif et géométrique pour le premier et basé sur l'optimisation sous contrainte pour le deuxième. Ces deux approches sont par ailleurs directement disponibles dans la bibliothèque logicielle de navigation `Nav2` [9] de ROS 2, sous la forme de modules. C'est cela qui en a fait des candidats pour le contrôle de la plateforme réelle et qui a facilité leur intégration pour les experimentations.

Algorithme DWB

Principe de fonctionnement

L'approche par fenêtre dynamique (Dynamic Window Approach (DWA)), introduite par Fox *et al.* [10], est une méthode de planification locale opérant directement dans l'espace des commandes du robot, plutôt que dans l'espace des positions.

Le contrôleur DWB implémenté dans `Nav2`, reprend ce principe en proposant une architecture modulaire dans laquelle la fonction de coût est décomposée en plusieurs critères, appelés *critics*, pouvant être ajustés indépendamment.

À partir de l'état courant $\mathbf{x}_t$, une fenêtre dynamique $\mathcal{V}_t$ est d'abord construite afin de déterminer les commandes admissibles $\mathbf{u} \in \mathcal{V}_t$. Chaque commande est ensuite simulée sur l'horizon T_{sim} considéré et sa trajectoire est évaluée par les différents critères de coût J_i. Les commandes conduisant à une collision sont écartées, tandis que les autres reçoivent un coût pondéré. La commande présentant le coût minimal (ou maximal selon la convention choisie) $\mathbf{u}^*$ est finalement sélectionnée comme commande à appliquer au robot. Le fonctionnement général du contrôleur DWB est résumé dans l'algorithme 1, et une illustration de principe est donnée à voir en figure 2.2

Algorithme 1 Dynamic Window approach B (DWB)

Entrées : $\mathbf{x}_t$, $\mathbf{x}_g$, T_c, T_{sim}
Sortie : $\mathbf{u}^* = (v^*, \omega^*)_{t+1}$

$\mathcal{V}_t \leftarrow \textsc{DynamicWindow}(\mathbf{x}_t, T_c)$
for all $\mathbf{u} = (v, \omega) \in \mathcal{V}_t$ **do**
    $J(\mathbf{u}) \leftarrow +\infty$
    $\tau_{\mathbf{u}} \leftarrow \textsc{Simulate}(\mathbf{x}_t, \mathbf{u}, T_{\text{sim}})$
    if not $\textsc{Collision}(\tau_{\mathbf{u}})$ **then**
        $J(\mathbf{u}) \leftarrow \sum_{i \in \mathcal{C}} w_i \, J_i(\tau_{\mathbf{u}}, \mathbf{x}_g)$
    end if
end for
$\mathbf{u}^* \leftarrow \arg\min_{\mathbf{u} \in \mathcal{V}_t} J(\mathbf{u})$
return $\mathbf{u}^*$

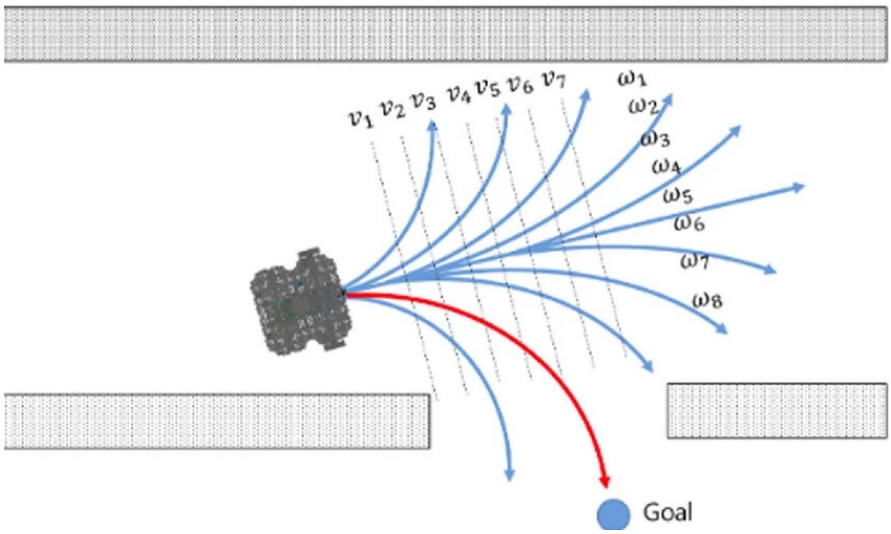


FIGURE 2.2 – Principe résumé du DWB, en bleu les trajectoires issues des commandes admissibles, en rouge la trajectoire retenue. Issue de [11]

Modèle cinématique du *rollout*

Le générateur de trajectoires (*rollout*) du contrôleur DWB intègre chaque commande candidate à l'aide d'un modèle cinématique de corps rigide plan, imposé par l'implémentation de `nav2_dwb_controller` et non substituable par un modèle personnalisé sans réécriture du module de génération de trajectoires [12]. Pour une base non-holonome telle qu'un robot différentiel ou à direction par dérapage, ce modèle se réduit exactement au modèle différentiel de l'équation 2.1, exprimé en variables de commande (v, ω) plutôt qu'en vitesses de roues (v_l, v_r).

Les seuls leviers alors exposés à l'utilisateur pour adapter le comportement du DWB à un robot à direction par dérapage sont par conséquent :

- L'espace des commandes admissible $\mathcal{V}$, via les bornes de vitesse et d'accélération, qui peuvent être resserrées empiriquement pour tenir compte, de façon non explicite et indirecte, du comportement en glissement du robot
- La fonction de coût, via le réglage des différents *critics*, qui peut pénaliser a posteriori les trajectoires les plus exposées au glissement (par exemple en défavorisant les rayons de courbure faibles) sans toutefois corriger l'erreur de prédiction du *rollout* lui-même.

Cette restriction constitue une limite intrinsèque du DWB pour une application à direction par dérapage et sera comparée au MPC, où le modèle cinématique employé dans la prédiction devient au contraire un paramètre explicite de la formulation.

Espace des vitesses admissibles

L'espace des vitesses effectivement explorées par l'algorithme, appelé fenêtre dynamique $\mathcal{V}$, résulte de l'intersection de trois sous-ensembles :

- $\mathcal{V}_s$: les vitesses admissibles imposées par les limites cinématiques des actionneurs ;
- $\mathcal{V}_a$: les vitesses atteignables compte tenu des accélérations maximales $\dot{v}_{\max}$, $\dot{\omega}_{\max}$ sur l'intervalle T_c ;
- $\mathcal{V}_r$: les vitesses admissibles garantissant que le robot puisse s'arrêter avant d'entrer en collision avec l'obstacle le plus proche, où $\mathrm{dist}(v,\omega)$ désigne la distance restante jusqu'à l'obstacle le long de l'arc de cercle associé à (v,ω).

De telle sorte que l'on ait les équations 2.11

$$\begin{cases} \mathcal{V} = \mathcal{V}_s \cap \mathcal{V}_a \cap \mathcal{V}_r, \\ \mathcal{V}_s = \{(v,\omega) \mid v_{\min} \le v \le v_{\max},\ \omega_{\min} \le \omega \le \omega_{\max}\}, \\ \mathcal{V}_a = \{(v,\omega) \mid v \in [v_c - \dot{v}_{\max} T_c,\ v_c + \dot{v}_{\max} T_c],\ \omega \in [\omega_c - \dot{\omega}_{\max} T_c,\ \omega_c + \dot{\omega}_{\max} T_c]\}, \\ \mathcal{V}_r = \left\{(v,\omega) \middle| v \le \sqrt{2\,\mathrm{dist}(v,\omega)\,\dot{v}_{\max}},\ \omega \le \sqrt{2\,\mathrm{dist}(v,\omega)\,\dot{\omega}_{\max}}\right\}. \end{cases} \tag{2.11}$$

Il est possible de rajouter un dernier terme à cet ensemble de vitesse admissibles afin d'y rajouter une condition de vitesse latérale nulle, puisque celle-ci n'est en principe pas directement comandable par un robot tel que le Warthog.

Fonction de coût

Chaque trajectoire admissible est évaluée à l'aide d'une fonction de coût de la forme

$$J(v,\omega) = w_1 \cdot \underbrace{\mathrm{heading}(\tau_{\mathbf{u}}, \mathbf{x}_g)}_{J_1} + w_2 \cdot \underbrace{\mathrm{dist}(\tau_{\mathbf{u}}, \mathbf{x}_g)}_{J_2} + w_3 \cdot \underbrace{\mathrm{vel}(\tau_{\mathbf{u}}, \mathbf{x}_g)}_{J_3} + \sum\nolimits_{i \in \mathcal{C} \setminus \{1,2,3\}} w_i\, J_i(\tau_{\mathbf{u}}, \mathbf{x}_g) \tag{2.12}$$

où $\mathrm{heading}(\cdot)$ mesure l'alignement de l'orientation finale du robot avec l'objectif ou le chemin global, $\mathrm{dist}(\cdot)$ la proximité aux obstacles, $\mathrm{vel}(\cdot)$ favorise les vitesses linéaires élevées, et les termes additionnels J_i correspondent aux critères supplémentaires configurables dans `nav2_dwb_controller`.
Les coefficients w_i jouent un rôle déterminant dans le comportement résultant et nécessitent un réglage spécifique à la plateforme et à l'environnement.

Intérêts et limites

L'intérêt principal du DWB réside dans sa simplicité algorithmique et son très faible coût de calcul, qui en font un algorithme adapté à une exécution temps réel embarqué à haute fréquence, notamment sur des robots à ressources limitées. Sa formulation dans l'espace dans des commandes garantit intrinsèquement l'admissibilité dynamique des trajectoires générées (à condition de disposer de ces valeurs pour la plateforme étudiée) Enfin, le fait qu'il s'agisse d'un des plus anciens algorithmes de navigation locale, largement déployé en robotique mobile, avec une architecture modulaire et reconfigurable, en fait une référence qu'il est nécessaire d'envisager.

Les principales limites du DWB dans le contexte d'un robot à direction par dérapage sont les suivantes :

- **caractère purement réactif et myope** : l'horizon de simulation T_{sim} étant court, l'algorithme est sujet aux minima locaux et ne construit aucune anticipation à long terme du comportement du robot

- **inadéquation du modèle cinématique aux effets de glissement** : l'hypothèse de trajectoires circulaires exactes (équation (2.1)) est mise en défaut par la cinématique réelle de la direction par dérapage, où le glissement induit un écart entre la trajectoire commandée et la trajectoire effectivement suivie
- **sensibilité au réglage des poids** w_i, dont le choix conditionne fortement la qualité du comportement obtenu et nécessite un ajustement empirique, peu généralisable d'une plateforme ou d'un environnement à l'autre.

Le Model Predictive Control (MPC)

Principe de fonctionnement

Le contrôle par modèle prédictif (Model Predictive Control (MPC)) est une stratégie de commande optimale, fondée sur la résolution répétée d'un problème d'optimisation sous contrainte sur un horizon fini glissant, à partir d'un modèle explicite du système commandé [13]. Contrairement au DWB, qui évalue un ensemble fini de trajectoires simulées, le MPC recherche à chaque instant la séquence optimale de commandes futures minimisant un critère de performance sur l'ensemble de l'horizon de prédiction, sous réserve du respect explicite de contraintes sur les états et les commandes. Seule la première commande de la séquence optimisée est appliquée au système, l'horizon est ensuite décalé et le problème est résolu à nouveau. Les solutions calculées au temps précédent peuvent être réutilisées pour faciliter la résolution du problème au temps suivant ou jetée. C'est cela qui confère au MPC sa robustesse vis-à-vis des perturbations et des erreurs de modèles. Le fonctionnement général du contrôleur MPC est résumé dans l'algorithme 2.

Algorithme 2 Model Predictive Control (MPC)

Entrées : $\mathbf{x}_t$, $\mathbf{X}_t^{\text{ref}}$, T_{sim}
Sortie : $\mathbf{u}_t^*$
1 : $\mathbf{U}_t^* \leftarrow \textsc{SolveMPC}\big(\mathbf{x}_t, \mathbf{X}_t^{\text{ref}}, T_{\text{sim}}\big)$ ▷ résolution de (2.14)
2 : $\mathbf{u}_t^* \leftarrow \mathbf{U}_t^*[0]$
3 : **return** $\mathbf{u}_t^*$

Modèle de prédiction

Le MPC repose sur un modèle prédictif explicite $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$, discrétisé sur la période d'échantillonnage T_s, où $\mathbf{x}_t$ et $\mathbf{u}_t$ reprennent les notations de la section 2.1, avec l'état $\mathbf{x}_k = [x, y, \theta]^T$ (éventuellement étendu aux vitesses locales $[v_x, v_y, \omega_z]^T$ pour un modèle dynamique) et commande $\mathbf{u}_t$ exprimée soit en vitesses de roue $[\omega_l, \omega_r]^T$, soit en couples moteurs $[\tau_L, \tau_R]^T$, selon le modèle retenu.

À la différence du DWB, dont le modèle de rollout est imposé par l'implémentation du module et se réduit à la cinématique différentielle idéale sous hypothèse de roulement sans glissement (équation 2.1), le modèle $f(\cdot)$ du MPC constitue un paramètre explicite et libre de la formulation. Cette propriété permet d'intégrer directement l'un des modèles de glissement présenté en section 2.1. Le choix de $f(\cdot)$ constitue alors un compromis entre fidélité de prédiction et coût de calcul pour la fréquence de commande visée.

De plus il faut ajouter aux incertitudes du modèle une mesure ou estimation fiable de l'état courant $\mathbf{x}_t$ à chaque instant (souvent par fusion odométrie/IMU/localisation), qui est complexe à obtenir en pratique sur les robots à direction par dérapage, comme on le verra dans le chapitre 5.

Formulation du problème d'optimisation

À chaque instant t, le MPC recherche, sur un horizon de prédiction T_{sim}, la séquence optimale de commandes $\mathbf{U}_t$ permettant de suivre au mieux une trajectoire de référence $\mathbf{X}_t^{\text{ref}}$, définies dans équations 2.13

$$\mathbf{U}_t = \begin{bmatrix} \mathbf{u}_{t\,|\,t}^T & \mathbf{u}_{t+1\,|\,t}^T & \cdots & \mathbf{u}_{t+T_{\text{sim}}-1\,|\,t}^T \end{bmatrix}^T, \quad \mathbf{X}_t^{\text{ref}} = \begin{bmatrix} (\mathbf{x}_t^{\text{ref}})^T & (\mathbf{x}_{t+1}^{\text{ref}})^T & \cdots & (\mathbf{x}_{t+T_{\text{sim}}}^{\text{ref}})^T \end{bmatrix}^T, \tag{2.13}$$

où $\mathbf{u}_{i|t}$ désigne la commande prédite à l'instant i à partir de la mesure disponible en t, et $\mathbf{x}_i^{\text{ref}}$ l'état de référence visé au même instant, issu de la trajectoire ou de l'objectif à atteindre.
Le problème d'optimisation résolu à chaque instant t s'écrit alors sous la forme des équations 2.14

$$\begin{cases} \mathbf{U}_t^* = \arg\min_{\mathbf{U}_t} J\Big(\mathbf{x}_{t|t}, \mathbf{U}_t, \mathbf{X}_t^{\text{ref}}\Big), & \\ \text{s.c.} \quad \mathbf{x}_{i+1|t} = f\big(\mathbf{x}_{i|t}, \mathbf{u}_{i|t}\big), & i \in [\![t,\ t+T_{\text{sim}}-1]\!], \\ \mathbf{u}_{i|t} \in \mathcal{V}, & i \in [\![t,\ t+T_{\text{sim}}-1]\!], \\ \mathbf{u}_{i|t} - \mathbf{u}_{i-1|t} \in [\Delta\mathbf{u}_{\min}, \Delta\mathbf{u}_{\max}], & i \in [\![t+1,\ t+T_{\text{sim}}-1]\!], \\ \mathbf{x}_{i|t} \in \mathcal{X}_{\text{adm}}, & i \in [\![t,\ t+T_{\text{sim}}]\!], \\ \mathbf{x}_{t|t} = \mathbf{x}_t. & \end{cases} \tag{2.14}$$

On observe un parallèle avec le DWB : là où la fenêtre dynamique $\mathcal{V} = \mathcal{V}_s \cap \mathcal{V}_a \cap \mathcal{V}_r$ (équation 2.11) résulte de l'intersection de trois sous-ensembles restreignant l'échantillonnage d'un seul pas de commande, le MPC exprime ces trois restrictions comme des contraintes distinctes sur l'ensemble de l'horizon T_{sim}. $\mathcal{V}$ noue ici le rôle de $\mathcal{V}_s$ qui borne les commandes admissibles imposées par les actionneurs, la contrainte sur $\Delta\mathbf{u}$ joue le rôle de $\mathcal{V}_a$ qui limite les variations de commande d'un pas à l'autre, et la contrainte d'état $\mathcal{X}_{adm}$ généralise $\mathcal{V}_r$ en portant directement sur la position future plutôt que sur la vitesse.

Lorsque le modèle $f(\cdot)$ retenu et les contraintes sont linéaires et le critère J quadratique, le problème 2.14 se ramène à un programme quadratique résolu efficacement en ligne. Dans le cas d'un modèle non linéaire tel que 2.7 ou 2.8, on parle de commande prédictive non linéaire (Nonlinear MPC (NMPC)), résolue avec des solveurs d'optimisation non linéaire comme dans ce papier [14]. Dans les deux cas, le critère J repose sur des matrices de pondération $\mathbf{Q}$, $\mathbf{R}$ et $\mathbf{P}$ détaillées dans la section suivant, jouant un rôle analogue à celui des coefficients w_i du DWB (équation 2.12).

Fonction de coût

Le critère J introduit dans le problème 2.14 s'exprime, dans sa forme quadratique la plus courante, par l'équation 2.15 :

$$J\Big(\mathbf{x}_{t|t}, \mathbf{U}_t, \mathbf{X}_t^{\text{ref}}\Big) = \sum_{i=1}^{T_{\text{sim}}-1} \Big(\underbrace{\|\mathbf{x}_{t+i|t} - \mathbf{x}_{t+i}^{\text{ref}}\|_{\mathbf{Q}}^2}_{\text{écart à la référence}} + \underbrace{\|\mathbf{u}_{t+i|t}\|_{\mathbf{R}}^2}_{\text{effort de commande}} \Big) + \underbrace{\|\mathbf{x}_{T_{\text{sim}}|t} - \mathbf{x}_{T_{\text{sim}}}^{\text{ref}}\|_{\mathbf{P}}^2}_{\text{coût terminal}}, \tag{2.15}$$

où $\|\mathbf{x}\|_{\mathbf{M}}^2 = \mathbf{x}^T\mathbf{M}\mathbf{x}$ désigne la norme quadratique pondérée par une matrice $\mathbf{M}$ définie positive. Cette fonction de coût repose ainsi sur trois matrices de pondérations :

- $\mathbf{Q}$, qui pénalise à chaque instant $t+i$ l'écart entre l'état prédit $\mathbf{x}_{t+i|t}$ et l'état de référence $\mathbf{x}_{t+i}^{\text{ref}}$
- $\mathbf{R}$, qui pénalise l'amplitude de la commande prédite $\mathbf{u}_{t+i|t}$, permettant de limiter la consommation énergétique ou l'agressivité des manoeuvres
- $\mathbf{P}$, le coût terminal, qui pondère spécifiquement l'écart à l'état de référence en fin d'horizon $\mathbf{x}_{T_{\text{sim}}|t}$

À la différence des coefficient w_i du DWB, qui pondèrent des critères géométriques hétérogènes J_i évalués a posteriori sur une trajectoire déjà simulée, les matrices $\mathbf{Q}$, $\mathbf{R}$ et $\mathbf{P}$ interviennent directement dans le problème d'optimisation résolu à chaque itération. Le réglage de ces matrices, tout comme le choix de l'horizon T_{sim}, n'est pas simple et conditionne fortement le compromis entre réactivité, efforts commandés et stabilité, et reste largement empirique [13].

Le contrôleur Model Predictive Path Integral (MPPI)

Au sein de `Nav2`, le contrôleur `nav2_mppi_controller` constitue une implémentation de commande prédictive fondée sur l'intégrale de chemin (MPPI), une variante stochastique et fondée sur l'échantillonnage

du MPC. Il s'affranchit de la nécessité de différentiabilité du modèle et de la fonction de coût, au prix d'un besoin en ressources de calcul (idéalement parallélisées sur GPU) plus important encore que pour un MPC classique.

Bien que ce contrôleur soit disponible sur `Nav2` et qu'il présente de nombreux intérêts, sa complexité de calcul ainsi que le fait que nous avions déjà sélectionné deux algorithmes de contrôle ń classiques ż nous ont détournés de son usage. Nous le mentionnons toutefois ici par souci d'exhaustivité.

2.3 Limites des approches classiques

Les différentes modélisations présentées dans ce chapitre permettent d'approcher la cinématique et la dynamique d'un robot à direction par dérapage. Toutefois, malgré les simplifications proposées, la dynamique réelle du robot reste fortement dépendante de son interaction avec le sol. Les variations d'état de surface et de nature du sol peuvent ainsi modifier significativement le comportement du robot, notamment du fait des phénomènes de glissement. Une modélisation suffisamment précise de ces interactions nécessiterait alors de prendre en compte des phénomènes relevant de la terramécanique, ce qui complexifie fortement le modèle et limite son utilisation dans un contexte de contrôle en temps réel.

Ces problématiques sont également présentes lors de l'utilisation de contrôleurs classiques pour la navigation. Les approches DWB et MPC génèrent en effet des commandes en s'appuyant sur une représentation explicite du comportement du robot et de son environnement, dont les limites ont été détaillées précédemment : le caractère réactif et myope du DWB, sujet aux minima locaux et à une sensibilité forte au réglage empirique de ses poids, ainsi que la dépendance du MPC à la fidélité de son modèle de prédiction et son coût de calcul plus élevé, en particulier en formulation non linéaire.

Ces observations soulèvent alors la question de l'intérêt d'une approche ne nécessitant pas de disposer a priori d'une modélisation précise du robot et permettant d'apprendre directement une politique de commande à partir de l'expérience. L'apprentissage par renforcement (Apprentissage par Renforcement (RL)) constitue une approche particulièrement intéressante dans ce contexte : plutôt que de modéliser explicitement l'ensemble des phénomènes physiques intervenant lors du déplacement, il est possible d'apprendre une politique permettant au robot de sélectionner ses actions en fonction de l'état observé.

Cette approche est d'autant plus pertinente dans le cadre de ce stage que les expérimentations ont été réalisées au sein de l'AMIAD, offrant un contexte favorable à l'étude de méthodes fondées sur l'IA. L'objectif de la suite de ce travail sera ainsi d'étudier la viabilité et l'intérêt d'une approche par apprentissage et de déterminer dans quelle mesure celle-ci peut répondre aux limitations identifiées avec les approches de modélisation et de contrôle classiques.

Chapitre 3

Simulation

Le chapitre précédent a présenté les modèles théoriques et les algorithmes de navigation classique retenus pour le contrôle du Warthog. Avant un déploiement sur le robot réel, il est nécessaire de disposer d'un environnement de test permettant de valider ces algorithmes et d'autres dans des conditions maîtrisées, répétables et sans risque matériel. Ce chapitre présente le travail de mise en place d'un jumeau numérique du robot dans la suite logiciel Isaac de NVIDIA (Isaac Sim et Isaac Lab), depuis la création du robot à la génération procédurale d'environnements de test.

3.1 Isaac Sim

Contexte et choix du simulateur

Dans le cadre du projet dans lequel s'inscrit ce stage, les robots disposent déjà d'une simulation sous Gazebo Harmonic [15] couplée à ROS 2. Ce logiciel bien qu'éprouvé et largement documenté dans l'écosystème ROS, présente deux limitations majeurs au regard des besoins pour les robots étudiés lors de ce stage :

- L'absence de rendu photoréaliste est pénalisant dès lors que l'on souhaite utiliser des algorithmes exploitant des capteurs caméra (segmentation, odométrie visuelle, ...) et non plus uniquement des capteurs de type LiDAR ou IMU
- L'architecture de simulation physique reposant essentiellement sur du calcul CPU limite la parallélisation d'environnements pourtant nécessaire pour des applications de génération de données ou d'apprentissage par renforcement

Pour répondre à ces deux contraintes a été retenu NVIDIA Isaac Sim [16]. Construit sur la plateforme Omniverse et le format d'environnements 3D ouvert Universal Scene Description (USD) [17], Isaac Sim s'appuie sur le moteur physique PhysX ainsi que sur un moteur de rendu Ray Tracing Texel eXtreme (RTX) permettant un rendu photoréaliste en temps réel, y compris pour la simulation de capteurs caméra (RGB, profondeur, segmentation sémantique, ...). Son architecture est en outre nativement pensée pour l'exécution vectorisée sur GPU, ce qui permet de faire tourner un grand nombre d'instances de simulation en parallèle, comme le montrent les travaux à l'origine de ce logiciel, Isaac Gym [18] et Isaac Lab [19].

En revanche Isaac Sim est un outil nettement plus lourd que Gazebo en termes de ressources matérielles, ce qui constitue un compromis assumé au regard des besoins du projet en matière de capteurs visuels et de parallélisation. Les capacités matérielles de l'ordinateur sur lequel ce travail a été réalisé est disponible en Annexe 6.

Principe de fonctionnement

Isaac Sim organise une scène de simulation sous la forme d'un *stage* USD, une structure de graphe de *prims* (primitives) hiérachiques dans laquelle chaque objet de la scène (corps rigides, articulation, capteurs, matériaux, lumières, ...) est représenté par un ou plusieurs *prims* dotés de *schemas* définissant leurs propriétés physiques visuelles. La simulation physique est déléguée au moteur PhysX, tandis que le rendu s'appuie

sur le moteur RTX de NVIDIA. L'intégration avec ROS 2 est assurée par une extension dédiée (*ROS 2*

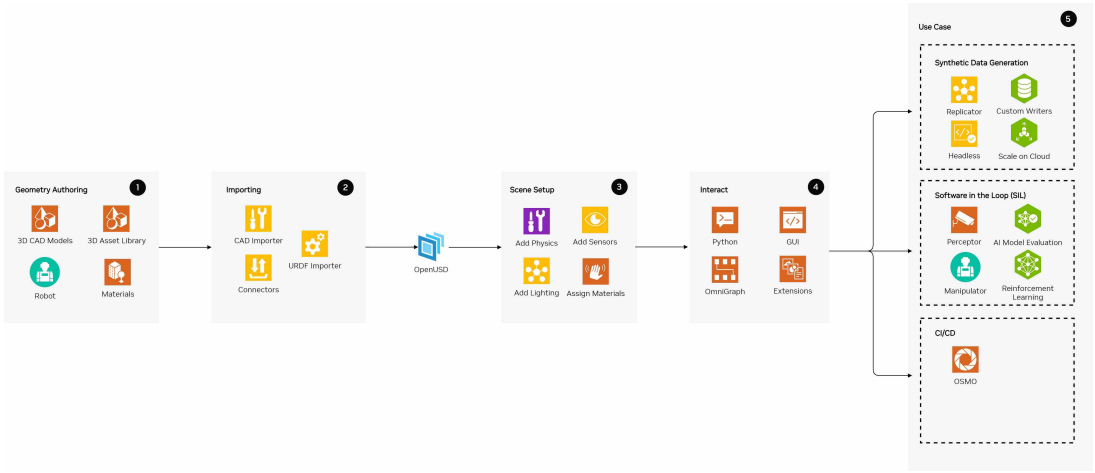


FIGURE 3.1 – Architecture logicielle de Isaac Sim

Source : Documentation Isaac Sim[16]

Bridge) et par l'outil *OmniGraph*, un système de programmation visuelle par graphes de noeuds permettrant de construire des publication et souscriptions de topics ROS 2 directement connectées aux données de simulation. L'architecture globale d'Isaac Sim est illustré dans la figure 3.1.

3.2 Import et configuration des URDF

Méthode manuelle via l'interface graphique

La première approche mise en oeuvre pour simuler le robot a consisté à importer le modèle Unified Robot Description Format (URDF) du robot (fourni par le constructeur) directement depuis l'interface graphique d'Isaac Sim. Cet import réalise automatiquement la conversion du modèle URDF en une hiérarchie USD, avec tout ce qui est nécessaire au solveur physique. Il faut néanmoins prendre soin de retirer les balises spécifiques à une simulation Gazebo puisqu'elles ne sont pas compatibles avec Isaac Sim et provoquent des plantages.

Une fois le module importé, les capteurs nécessaires aux algorithmes de navigation sont rajoutés manuellement sur les articulations correspondantes du robot.

La publication de l'ensemble des données des capteurs vers ROS 2 ainsi que la souscription des commandes de vitesse est ensuite configurée via des graphes *OmniGraph*, assemblés manuellement pour chaque chaque capteur et chaque topic faute de pouvoir le faire automatiquement. Les graphes pour chaque noeud ROS sont disponibles en Annexe 6.

Cette approche manuelle, si elle a permis une prise en main progressive du logiciel et une première validation de la chaîne de simulation, présente l'inconvénient d'être longue et peu reproductible dès que le modèle du robot évolue. Chaque modification nécessite de refaire tout ou partie de la configuration manuellement.

Migration vers Isaac Sim 6.0 et automatisation

L'arrivée de la version 6.0 d'Isaac Sim au cours du stage a permis une évolution importante pour le projet. Cette version au contraire de la 5.1 a aligné son API Python avec la distribution Python native de ROS 2 Jazzy. Cette convergence a permis de remplacer la configuration manuelle décrite précédemment par une chaîne automatique scriptée.

Un script Python a ainsi été développé pour automatiser l'ensemble du processus et est décrit dans l'algorithme 3

Algorithme 3 Conversion URDF vers USD avec ajout de capteurs

Entrées : Fichier URDF du robot
Sortie : Fichier USD du robot avec ses capteurs et OmniGraphs configurés
1 : Importer le fichier URDF
2 : Supprimer les balises `gazebo`
3 : Parser les balises `isaacsim`
4 : Supprimer les balises `isaacsim`
5 : Convertir le fichier URDF au format USD
6 : Ajouter les capteurs grace aux informations des balises `isaacsim`
7 : Configurer les *OmniGraphs* correspondants aux capteurs
8 : Configurer les *OmniGraphs* génériques (horloge, TF, commande)
9 : Retourner le fichier USD du robot

Cette automatisation permet en théorie l'utilisation de n'importe quel robot simulé sur Gazebo dans Isaac Sim à condition d'y rajouter des balises `isaacsim` supplémentaires, puisque le seul fichier URDF du robot permet son chargement dans Isaac Sim.

Il faut noter tout de même que ce script d'automatisation n'est pas complètement universel et pourrait bénéficier d'améliorations pour le rendre complètement compatible avec les balises `gazebo`. De plus certaines parties de celui-ci ont été codés en dur par simplicité.

Mise en route complète et intégration dans la stack logicielle

Afin que la simulation puisse être utilisée de manière transparente par la pile logicielle du robot réel, une mise en route (*bringup*) complète a été mise en place. Celle-ci a pour rôle que la simulation Isaac Sim se substitue au robot réel du point de la pile logicielle de navigation. Ainsi ce script Python permet la mise en route de la simulation et lance un conteneur *Docker* faisant tourner les mêmes noeuds que sur le robot réel, permettant de tester les algorithmes de navigation. La figure 3.2 permet de voir le Warthog dans un

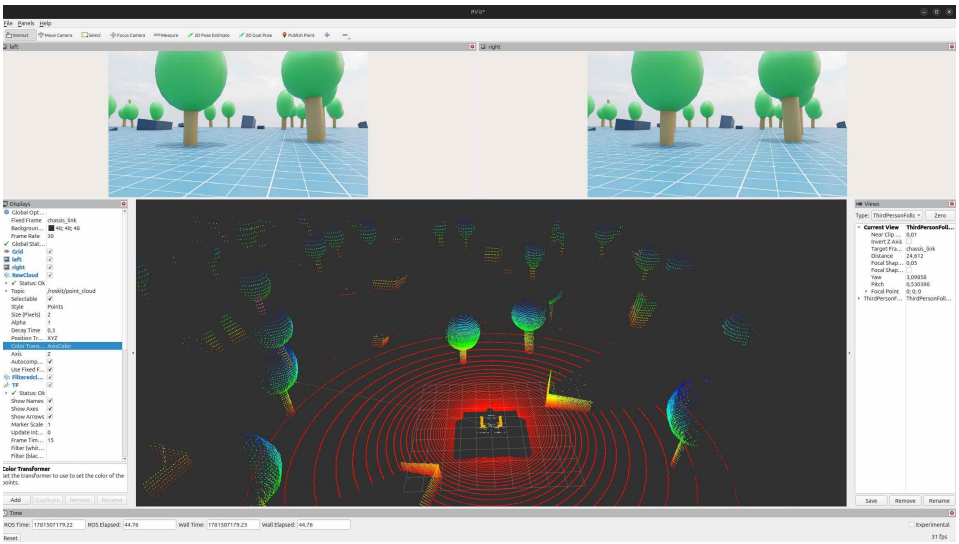


FIGURE 3.2 – Visualisation des capteurs du Warthog dans la Simulation avec RViz

environnement simulé et les données de ses capteurs captées par RViz et donc bien transmis via ROS 2.

Retour d'expérience sur la prise en main

Malgré une documentation officielle relativement fournie [16], la prise en main d'Isaac Sim s'est révélée plus difficile qu'anticipé. La taille très importante du projet (multiplicité des extensions, des API et des couches d'abstraction superposées entre Omniverse Kit, Isaac Sim et Isaac Lab) et les incohérences notables entre les versions successives (changement d'API, de nommage d'extensions ou de comportement par défaut), rendent certaines ressources en ligne obsolètes d'une version à l'autre (y compris de la documentation courante), rendant la prise en main du logiciel difficile.

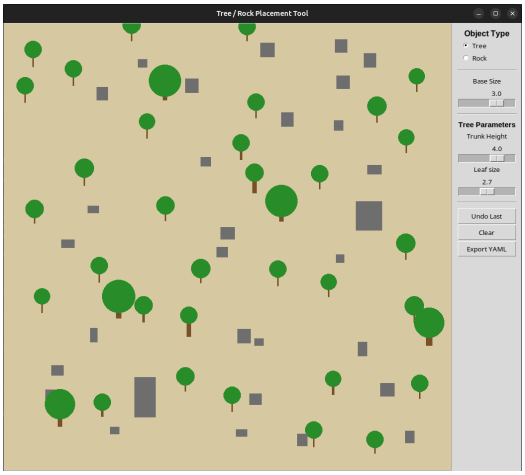
Il n'en reste néanmoins qu'Isaac Sim permet comme aucun autre simulateur de combiner à la fois un bon solveur physique et un très bon rendu graphique.

3.3 Génération procédurale d'environnements

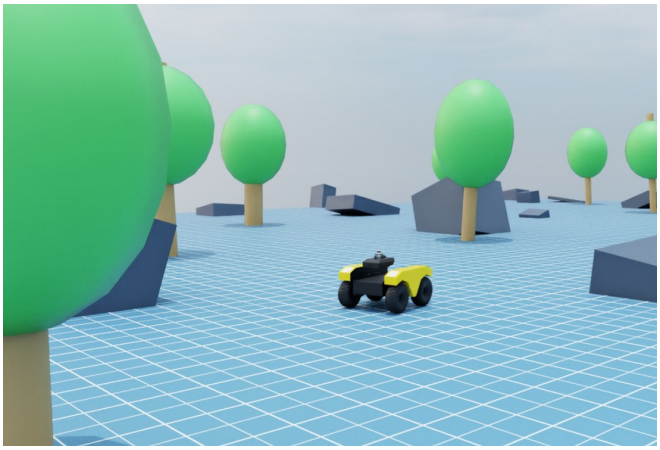
L'environnement plat par défaut fourni par Isaac Sim n'étant pas représentatif des conditions d'évolution réelles du robot, plusieurs algorithmes de génération procédurale de terrain ont été développées, avec un degré croissant de complexité.

Environnement simple avec obstacles

Un premier générateur, simple a été mis en place pour obtenir rapidement un environnement minimal. Il est constitué d'un sol plat, et d'un ensemble d'arbres et de rochers modélisés procéduralement à partir de primitives géométriques (cubes, cylindres et ellipsoïdes) orientés aléatoirement sur le terrain et positionnés à l'aide d'une interface graphique minimale, représentée en figure 3.3a. Ce générateur a permis d'obtenir une diversité visuelle minimale afin de tester le bon fonctionnement des capteurs et de la pile logicielle `Nav2` tel que cela est représenté sur les figures 3.2 et 3.3b.



(a) Interface graphique de placement des arbres et rochers



(b) Le Warthog dans l'environnement généré

FIGURE 3.3 – Placement d'arbre et de rochers

Prototype de génération de routes

Afin de préparer l'intégration ultérieure d'algorithmes de navigation exploitant des informations relatives à la nature du sol, nous avons étudié la génération de textures pour le terrain. Dans cette perspective, un second prototype a été développé afin d'explorer la génération automatique de routes sinueuses à l'aide de l'algorithme A* [20].

L'approche proposée consiste à modifier le coût associé aux déplacements lors de la recherche, afin de favoriser les trajectoires présentant davantage de variations spatiales plutôt que de rechercher le chemin

géométriquement le plus court. Pour cela, un terme supplémentaire basé sur un bruit de Perlin [21] est introduit dans le coût de déplacement.
La fonction d'évaluation de l'algorithme A* s'écrit alors :

$$f(n) = \underbrace{\sum_{i=1}^{k} [d(n_{i-1}, n_i) + \lambda P(n_i)]}_{g(n)} + \underbrace{d(n, n_{\text{objectif}})}_{h(n)}, \qquad n_k = n \tag{3.1}$$

où $d(n_{i-1}, n_i)$ représente la distance entre deux noeuds successifs, $P(n_i)$ la valeur du bruit de Perlin au noeud n_i, λ un paramètre contrôlant l'influence du bruit sur le coût de déplacement, $g(\cdot)$ le coût cumulé de déplacement et $h(\cdot)$ la fonction heuristique.

La génération de la carte représentant le tracé de la route a été mené à son terme, mais faute de temps, l'étape de conversion de cette carte 2D en objet 3D importable dans Isaac Sim n'a pas été réalisée. Ce prototype n'a pas été intégré dans le modèle de carte utilisé par la suite, mais constitue une piste d'amélioration identifié pour de futurs travaux.

Segmentation du sol par diagramme de Voronoï

Un troisième algorithme plus abouti, permet de générer un terrain segmenté en plusieurs classes de friction, chacune associée à un matériau physique distinct, afin de simuler un sol hétérogène. Le principe repose sur la génération d'un diagramme de Voronoï sur le domaine du terrain, dont les cellules sont ensuite triangulées pour produire un ensemble de maillages 3D discontinu, chacun correspondant à l'ensemble des cellule d'une classe de friction donnée.

Des obstacles simples (cubes et cylindres) sont ensuite placés aléatoirement sur ce terrain. L'algorithme renvoie, en plus d'un fichier contenant les objets 3D, la liste des positions et boîtes englobantes de ces obstacles, ce qui permet d'efficacement éviter l'apparition du robot ou de sa cible sur un obstacle lors de l'initialisation de la simulation.

C'est cet algorithme qui est utilisé créer les terrains des entraînements par RL de la section 5.2.

Génération avancée avec relief

Un quatrième algorithme, le plus abouti, enrichit l'approche précédente en y ajoutant du relief. Le diagramme de Voronoï utilisé pour la segmentation en classes de frictions est conservé, mais la génération géométrique du terrain ne repose plus sur une triangulation de chaque cellule, mais plutôt sur une carte d'élévation générée à une résolution donnée. L'élévation de la carte est determinée par un bruit de Perlin superposé au diagramme de Voronoï.

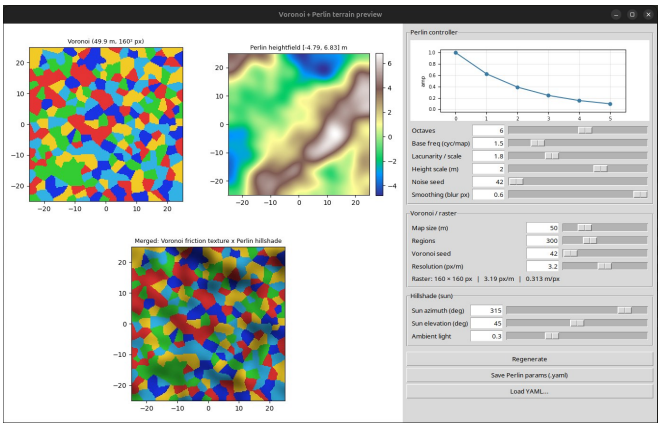
Cette approche permet d'obtenir un terrain à la fois irrégulier en relief et segmenté en différentes zones de friction, tout en conservant les mécanismes de placement d'obstacles.

Une interface visuelle a aussi été développée pour faciliter la création d'environnements à l'aide de cet algorithme et est représenté en figure 3.4a, et un exemple de génération est présenté en figure 3.4b.

3.4 Adaptation et ajouts pour Isaac Lab

La simulation présentée jusqu'ici a permis de valider la construction d'un jumeau numérique du Warthog tant du point de vue du comportement physique que logiciel. Cependant dans la perspective d'implémenter des contrôleurs basés sur l'apprentissage par renforcement, un certain nombre d'adaptations sont nécessaires.

Isaac Lab [19], le logiciel de création et d'entraînement de modèles d'IA pour la robotique de NVIDIA présente de nombreuses fonctionnalités communes avec Isaac Sim. Construit sur le même socle logiciel, il présente les atouts présentés en section 3.1. Néanmoins, quelques subtilités existent afin d'adapter la simulation qui avait été préparée pour Isaac Sim à Isaac Lab. Veuillez noter en outre que la version du



(a) Interface graphique de génération de la carte d'élévation

(b) Un environnement généré avec cet algorithme

FIGURE 3.4 – Environnement généré procéduralement avec relief et classes de frictions. La carte obtenue est directement issue des l'aperçu de l'interface graphique

logiciel utilisé ici est la V3.0.0 beta 2, un détail important dans la mesure où une version ultérieure rendra peut-être les adaptations présentées superflues.

Tout d'abord, toute la pile ROS 2 est écarté pour l'entraînement du robot. En effet l'ensemble des données utiles (physique du robot et capteurs) peuvent être obtenus directement dans le logiciel. De plus l'usage de ROS 2 pour la communication entre les processus ralentirait considérablement l'entraînement d'un modèle d'IA.

Ensuite, de nouveaux capteurs virtuels ont été ajoutés au robot afin de préparer les entrées du modèle d'IA développé en section 4.2. Les choix spécifiques de ces capteurs sont justifiées en section 4.3.

Détection de collisions

Un premier capteur permet d'implémenter un système de détection de collision afin de pénaliser tout comportement du robot le faisant entrer en contact avec des obstacles.

Pour se faire, quatre plaques sont disposées autour du châssis du robot, tel qu'illustré en figure 3.5. Lorsqu'un effort est détectée entre n'importe quelle plaque et un obstacle, le signal est alors récupéré et interprété comme une collision.

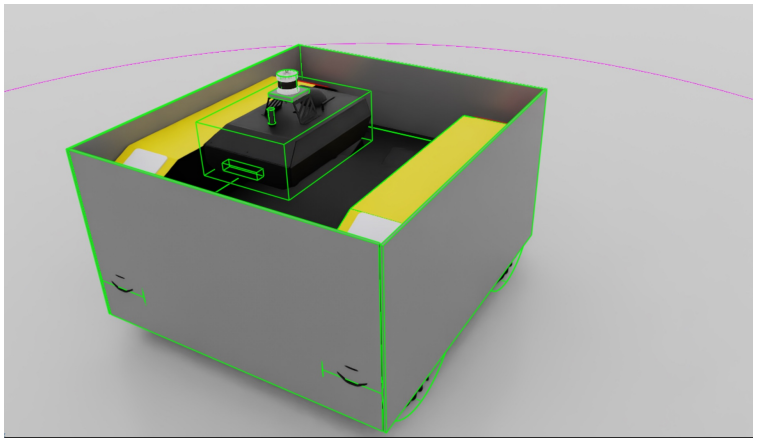


FIGURE 3.5 – Plaques de détection de collision entre le robot et les obstacles. Les lignes vertes représentent les boîtes de collision du robot.

Le système de plaques répond à une contrainte technique liée à l'API de détection de collision d'Isaac Lab, dont le fonctionnement dépend de l'architecture de l'arbre d'articulations du robot. De surcroît cette astuce permet d'éviter de comparer la collisions entre l'objet obstacle (l'ensemble des obstacles ayant été réunis dans un seul objet 3D) et toutes les primitives du robot, mais seulement avec chacune des plaques, améliorant ainsi aussi les performances de la simulation.

Génération d'une carte de coût

Un second capteur permet de produite une carte de coût (*costmap*), c'est à dire un tableau représentant l'environnement autour du robot, et contenant des valeurs représentant différentes propriétés du terrain (type, difficulté de déplacement, dangerosité, obstacles). Ce capteur virtuel spécifique à la simulation permet d'abstraire la chaîne de traitement (LiDAR, caméra) qui produit les informations sémantiques de l'environnement proche du robot, sous la forme d'une carte de coût. Dans une chaîne de contrôle classique la carte de coût est produite par Nav 2 à partir de données capteurs et peut être enrichie par l'utilisateur. Son utilisation ici pour entraîner un agent par RL a pour but de voir si, avec des informations équivalentes à un contrôleur classique, un contrôleur basé sur le ML peut produire un comportement équivalent ou meilleur que celui-ci.

Comme il n'existe actuellement dans Isaac Lab aucun capteur permettant de générer une carte de coût de l'environnement, nous devons en construire un.

Pour cela nous exploitons le `RayCaster` du logiciel : ce capteur natif d'Isaac Lab permet d'émettre une multitude de rayons et renvoie la distance parcourue par chacun d'eux jusqu'au premier obstacle rencontré. Afin de pouvoir l'exploiter, nous devons modifier la génération du terrain en rajoutant, sous la carte, une représentation alternative de l'environnement en relief, où chaque niveau de profondeur correspond à une propriété du terrain. Le `RayCaster` placé sous la carte permet alors de constuire la carte de coût en comparant les mesures avec la base de données générée. Le principe de fonctionnement de ce capteur est résumé dans

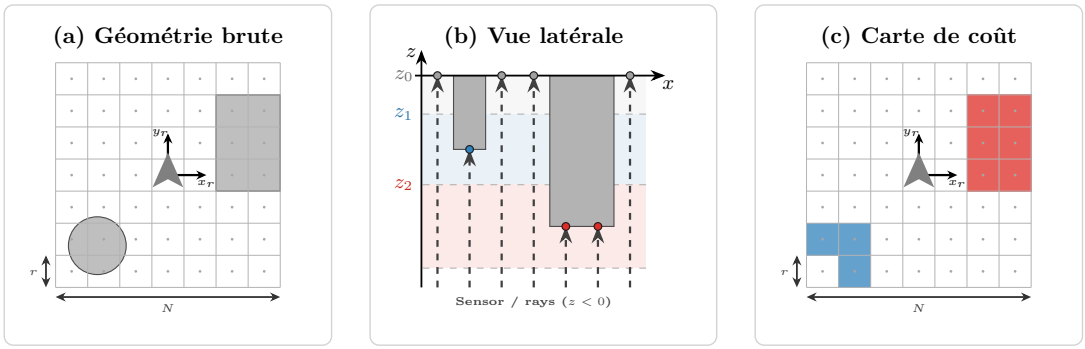


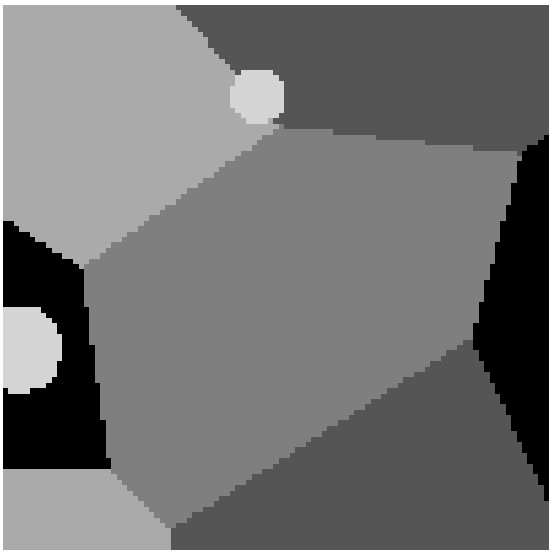
FIGURE 3.6 – Principe de fonctionnement du capteur de carte de coût

le diagramme 3.6. La figure 3.7 présente la carte de coût obtenue 3.7a ainsi que l'environnement où elle a été généré 3.7b.

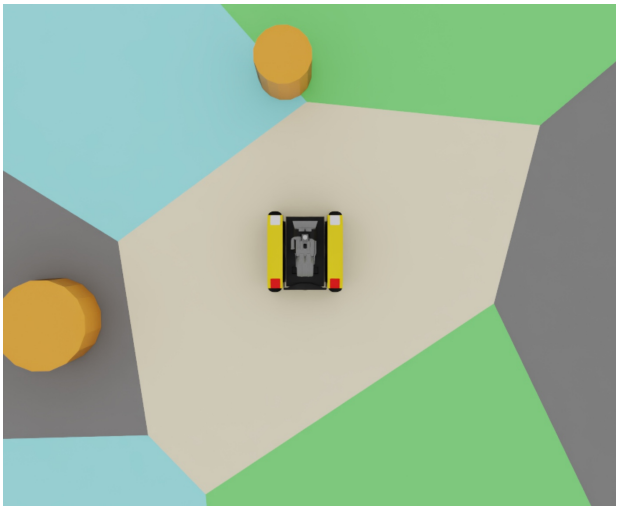
Bruitage de la carte de coût

Une dernière étape prototypée mais non implémentée dans Isaac Lab permet de perturber les frontières de la carte de coût sémantique.

Les quatre méthodes proposées ci-après suivent une logique d'optimisations successives afin de permettre leur utilisation dans la boucle d'entraînement du contrôleur par RL. Elles consistent toutes à déterminer une zone de perturbation autour des frontières du masque, à générer un bruit aléatoire, à le pondérer spatialement, puis à le lisser et à seuiller le résultat. Les différences résident principalement dans la manière de calculer le champ de distance D et le champ de variance $\mathcal{V}$.



(a) carte de coût générée



(b) Environnement d'où a été généré la carte de coût

FIGURE 3.7 – Carte de coût et l'environnement dont elle est issue

La première méthode, la plus coûteuse, s'écarte du schéma additif des trois suivantes : elle combine un tirage stochastique seuillé par $\mathcal{V}$ à des opérations morphologiques de croissance et d'érosion, plutôt qu'une injection de bruit continu. Ces opération sont résumés par l'algorithme 4.

Algorithme 4 Méthode 1 : bruitage par tirage stochastique et morphologie

Entrées : M, λ, T, γ_g, γ_e
Sortie : M^*

$D \leftarrow \mathrm{DT}(\neg M) - \mathrm{DT}(M)$
$\mathcal{V} \leftarrow \exp\left(-\frac{|D|}{\lambda}\right)$
$F \leftarrow M$
for $i = 1$ à T **do**
    $U_i \leftarrow$ GENERATEUNIFORM()
    $\mathrm{Grow}_i \leftarrow U_i < \gamma_g \mathcal{V}$
    $\mathrm{Erode}_i \leftarrow U_i < \gamma_e \mathcal{V}$
    $F \leftarrow (F \cup \mathrm{Grow}_i) \setminus \mathrm{Erode}_i$
    $F \leftarrow \mathrm{Erode}\big(G_\sigma * F > \tau\big)$
end for
$M^* \leftarrow F$
return M^*

Les méthodes 2 à 4 partagent le même schéma additif, résumé par l'algorithme 5. Elles distinguent par leur définition respective du champ de distance et du champ de variance $\mathcal{V}$ utilisés dans l'algorithme 5.

Algorithme 5 Algorithme générique de bruitage (méthodes 2 à 4)

Entrées : M, $\mathcal{D}$, $\mathcal{V}$, T, σ, α, τ
Sortie : M^*
$D \leftarrow \mathcal{D}(M)$
$V \leftarrow \mathcal{V}(D, M)$
$F_0 \leftarrow M$
for $i = 1$ à T **do**
    $N_i \leftarrow$ GenerateNoise()
    $F_i \leftarrow G_\sigma * (F_{i-1} + \alpha\, V \odot N_i)$
end for
$M^* \leftarrow \mathbb{1}_{F_T > \tau}$
return M^*

La méthode 2 reprend la distance signée euclidienne :

$$D = \mathrm{DT}(\neg M) - \mathrm{DT}(M), \qquad \mathcal{V} = \exp\left(-\frac{|D|}{\lambda}\right) \tag{3.2}$$

La méthode 3 approxime cette distance par la frontière morphologique du masque, normalisée par filtrage gaussien :

$$D = M \setminus \mathrm{Erode}(M), \qquad \mathcal{V} = \frac{G_{\sigma_2}(D)}{\max\big(G_{\sigma_2}(D)\big) + \epsilon} \tag{3.3}$$

La méthode 4 se passe de toute extraction explicite de frontière en exploitant directement le masque lissé S :

$$S = G_{\sigma_2}(M), \qquad \mathcal{V} = 4S(1 - S) \tag{3.4}$$

Toutes les méthodes se terminent par un seuillage $M^* = \mathbb{1}_{F_T > \tau}$, à l'exception de la première qui procède par érosion itérative plutôt que par seuillage final.
Une analyse rapide des performances des quatre méthodes (implémentées avec Numpy) en temps d'exécution moyen sur 100 essais permet de constater une nette amélioration des performances qui passent de 10 ms et 8 ms pour les méthodes 1 et 2, et à 3 ms pour les méthodes 3 et 4. De plus, méthode 4 a été conçue spécifiquement pour éviter toute opération nécessitant un transfert de données entre CPU et GPU permettant ainsi d'optimiser les performance encore plus lors du passage vers l'environnement d'entraînement d'Isaac Lab. La figure 3.8 illustre les effets de chaque méthode.

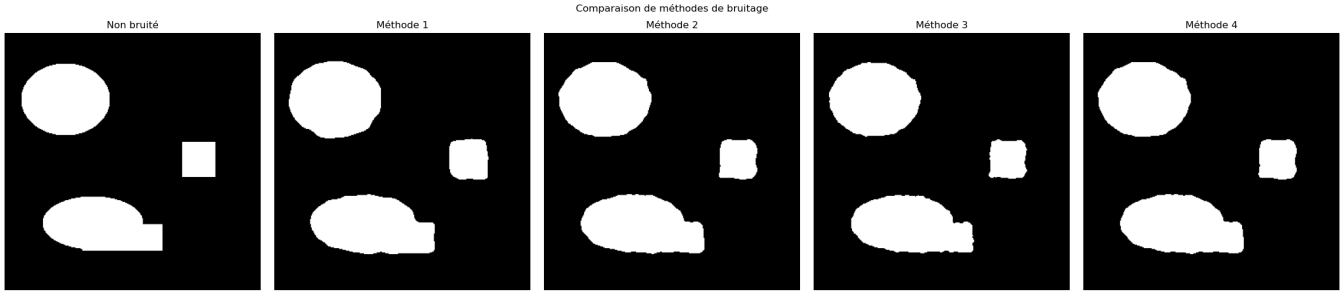


FIGURE 3.8 – Comparaisons d'algorithmes de bruitage de carte de coût

Chapitre 4

Apprentissage par renforcement

4.1 Théorie de l'apprentissage par renforcement

Éléments d'apprentissage automatique

L'apprentissage automatique (ML) est un champ d'étude de l'intelligence artificielle qui se fonde sur des approches mathématiques et statistiques pour donner à un ordinateur la capacité d'apprendre à partir de données. L'apprentissage supervisé est une tâche d'apprentissage automatique dont l'objectif est d'apprendre une fonction de décision à partir d'un ensemble d'exemples pour lesquels une sortie attendue est explicitement fournie. Cette formulation est adaptée aux approches classiques de perception ou de contrôle où chaque observation/action peut être associée à une annotation permettant de guider l'apprentissage. L'apprentissage par renforcement (RL) est une tâche d'apprentissage automatique qui se distingue de ce cadre par l'absence de vérité terrain associée à chaque décision. L'ordinateur, appelé *agent* apprend par interaction avec un environnement, en choisissant des actions et en recevant en retour un signal de récompense. L'objectif n'est plus de reproduire cible donnée, mais d'apprendre une stratégie permettant de maximiser la récompense cumulée au cours de l'interaction.

La quasi-totalité des méthodes de RL repose sur des réseaux de neurones artificiels qui permettent d'approcher n'importe quelle fonction [22]. Le perceptron multicouche (Multi-Layer Perceptron (MLP)) constitue l'architecture la plus simple d'un réseau de neuronnes, il s'agit d'un empilement de couches entièrement connectés, chacune appliquant une transformation affine suivie d'une fonction d'activation non linéaire (souvent ReLU ou tanh).

L'entraînement de ce type de réseau consiste à ajuster l'ensemble des paramètres $\Theta = \{\mathbf{W}, \mathbf{b}\}$ par descente de gradient stochastique, le gradient de cette fonction de coût étant calculé par rétropropagation.

Dans le cadre du RL, ce ne sont plus les paramètres qui minimisent une erreur de prédiction qui sont recherchés, mais ceux qui maximisent l'espérance de récompense cumulée obtenue par l'agent. C'est ce changement d'objectif qui structure le formalisme présenté dans la suite de cette section. Une schéma général de principe du RL est proposé en figure 4.1

Formalisme des processus de décision markoviens

Le cadre mathématique standard du RL est celui des processus de décision markoviens (Markov Decision Process (MDP)). Un MDP est défini par un quintuplet $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, où $\mathcal{S}$ est l'espace des états, $\mathcal{A}$ l'espace des actions, $P(s_{t+1} \mid s_t, a_t)$ la fonction de transition, $R(s_t, a_t)$ la fonction de récompense, et $\gamma \in [0, 1[$ le facteur d'actualisation.

Dans le cadre de ce travail, $\mathcal{S}$ regroupe les observations décrivant l'état du robot et de son environnement (position relative à l'objectif, dernière commande appliquée, éventuellement carte de coût locale), $\mathcal{A}$ correspond aux commandes de vitesse linéaire et angulaire envoyées au robot.

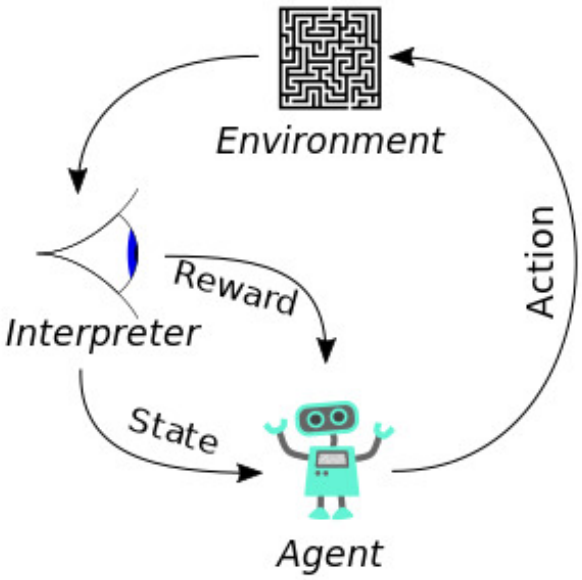


FIGURE 4.1 – Représentation simplifiée du principe de fonctionnement de l'RL. Issue de [23]

L'agent agit selon une politique $\pi_\Theta(a_t \mid s_t)$, paramétrée par un réseau de neurones de paramètres Θ, qui définit une distribution de probabilité sur les actions sachant l'état observé. L'objectif de l'apprentissage est de trouver les paramètres Θ^* qui maximisent l'espérance du gain cumulé (*cumulative return*)

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \tag{4.1}$$

sur les trajectoires issues de π_Θ. Deux fonctions de valeur sont usuellement définies pour quantifier la qualité d'un état ou d'une paire action-état pour une politique π :

$$V^\pi(s) = \mathbb{E}_\pi \left[G_t \mid s_t = s \right], \tag{4.2}$$

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[G_t \mid s_t = s,\ a_t = a \right]. \tag{4.3}$$

On définit également la fonction d'avantage $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$, qui mesure la qualité d'une action donnée par rapport au comportement moyen de la politique pour un état s donné. Cette quantité joue un rôle central dans les algorithmes acteur-critique présentés ci-après.

Grandes familles d'algorithmes

Les algorithmes de RL se répartissent schématiquement en trois familles.

Les méthodes à valeur (*value-based*), telles que le Q-Learning, apprennent une estimation de Q^π et en déduisent une politique *greedy* ou ϵ-*greedy*. Elles sont adaptées à des espaces d'action discrets et s'appliquent alors mal à la commande continue d'un robot mobile.

Les méthodes à politique (*policy-based*) optimisent directement les paramètres Θ de π_Θ par montée de gradient sur l'espérance gain cumulé, en s'appuyant sur le théorème du gradient de politique (*policy gradient theorem*) :

$$\nabla_\Theta J(\Theta) = \mathbb{E}_{\pi_\Theta} \left[\nabla_\Theta \log \pi_\Theta(a_t \mid s_t)\, A^{\pi_\Theta}(s_t, a_t) \right]. \tag{4.4}$$

Enfin, les méthodes acteur-critique combinent les deux approches. Un réseau *acteur* joue le rôle de la politique tandis qu'un réseau *critique* estime la fonction de valeur (ou d'avantage) utilisée pour réduire la variance de l'estimateur du gradient 4.4. C'est dans cette dernière famille que se situent trois algorithmes retenus pour comparaison dans le cadre de ce stage : PPO, SAC et A3C.

Proximal Policy Optimization (PPO) : PPO [24] est un algorithme acteur-critique *on-policy*, c'est à dire que les données utilisées pour chaque mise à jour des paramètres doivent avoir été générées par la politique courante. Pour éviter les mises à jour trop agressives qui déstabilisent l'apprentissage, PPO contraint le ratio entre la nouvelle politique et l'ancienne. Le caractère *on-policy* de PPO implique un besoin important de données récentes, ce qui en fait un algorithme particulièrement adapté à la simulation massivement parallèle discuté en section 4.3.

Soft Actor-Critic (SAC) : SAC [25] est, à l'inverse, un algorithme *off-policy*, il réutilise les transitions passées stockées dans une mémoire de rejeu (*replay buffer*), ce qui améliore son efficacité en pour un nombre d'échantillons donnée au prix d'une complexité d'implémentation accrue. Il introduit un terme de régularisation par entropie et un terme de température qui pondère l'incitation à l'exploration de nouvelles politiques. Cette formulation favorise des politiques plus stochastiques et robustes, mais rend l'entraînement plus coûteux en mémoire du fait de la mémoire de rejeu.

Asynchronous Advantage Actor-Critic (A3C) : A3C [26] repose sur l'exécution de plusieurs acteurs en parallèle, chacun interagissant avec sa propre copie de l'environnement et calculant des gradients locaux, qui sont ensuite appliqués de manière asynchrone à un réseau de paramètres partagés. Cette parallélisation, est historiquement pensée pour exploiter plusieurs coeurs CPU exécutant chacun une instance de simulation, et a été supplanté par les approches synchrones (A2C, PPO) lorsque la simulation est parallélisable sur GPU, ce qui est le cas dans l'environnement utilisé dans ce travail.

4.2 Choix du modèle et de l'architecture

Choix de l'algorithme d'apprentissage

Le choix s'est porté sur PPO plutôt que sur SAC ou A3C au regard des caractéristiques de l'environnement de simulation retenu. Ce dernier permet en effet d'exécuter plusieurs centaines d'environnements en parallèle sur GPU, générant ainsi un volume considérable de transitions à chaque pas de temps simulé. Cette caractéristique correspond au régime pour lequel PPO a été conçu. À l'inverse, l'intérêt principal de SAC, à savoir sa meilleure efficacité en nombre d'échantillons grâce à la mémoire de rejeu, perd une grande partie de sa pertinence lorsque la génération de données n'est plus le facteur limitant de l'entraînement. De plus, la possibilité de définir des épisodes courts par la répétition rapide des essais permis par la simulation parallèle, permet à l'agent PPO de recevoir un de signal de retour dense malgré l'absence de réutilisation des données. Faute de temps, la comparaison avec d'autres algorithmes, en particulier SAC, n'a pas pu être menée, bien qu'elle eût été pertinente.

Architecture de base : validation sur terrain plat

Rappelons que l'objectif de ce modèle de RL est de construire une politique pour un contrôleur bas niveau. On suppose alors qu'un planificateur haut niveau fournisse un point à atteindre dans le référentiel local du robot (la transformation du repère global au repère local étant effectué par la brique de localisation).

La première architecture développée est volontairement minimale, afin de valider la chaîne complète d'entraînement (définition des récompenses, de l'espace d'observation et d'action et l'intégration dans Isaac Lab) avant d'en complexifier progressivement les composantes. Il s'agit d'un MLP à deux couches cachées. Le vecteur d'observation fourni en entrée du réseau est composé de sept composantes :

- la distance euclidienne entre le robot et l'objectif à atteindre
- le cosinus de l'angle relatif entre l'orientation du robot et la direction de l'objectif
- le sinus du même angle
- la vitesse linéaire de la dernière commande appliquée
- la vitesse angulaire de la dernière commande appliquée
- le diamètre des roues du robot

- la largeur de l'essieu du robot

L'encodage de l'angle relatif sous la forme $(\cos, \sin)$ plutôt que par sa valeur brute permet d'éviter la discontinuité inhérente à une représentation angulaire directe (saut de π à $-\pi$), et facilite ainsi l'apprentissage par le réseau. L'inclusion de la commande précédemment appliquée permet de créer une boucle fermée sur la commande, afin d'anticiper l'ajout ultérieur d'un bruit sur l'action réellement exécutée par les moteurs. Il s'agissait également d'évaluer si le modèle parviendrait, à terme, à développer une forme d'odométrie implicite en mettant en relation les observations relatives à l'objectif et celles relatives à la commande précédente. L'inclusion du diamètre des roues et de la largeur du robot dans l'observation, bien que constante dans le monde réel, peut être *randomisée* afin de forcer l'agent à généraliser sa connaissance du modèle cinématique du robot plutôt qu'à en mémoriser un cas particulier. En pratique, les modèles entraînés ne faisaient pas varier ces grandeurs, pour des raisons de manque de temps dans l'implémentation de cette fonctionnalité.

En sortie, le réseau produit directement les deux commandes envoyées au robot : la vitesse linéaire d'avance v_x et la vitesse angulaire ω_z. Cette première architecture simple est représenté par le schéma de principe 4.2.

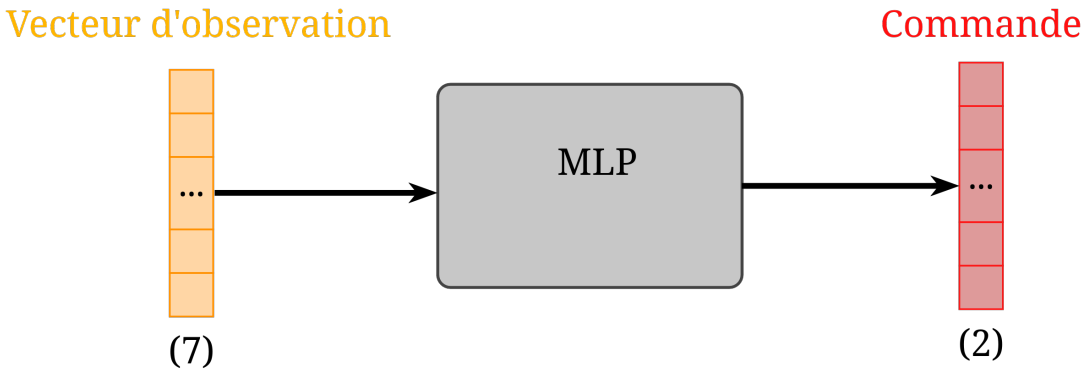


FIGURE 4.2 – Schéma simple de la première architecture de RL proposée

La fonction de récompense associée à cette première architecture combine plusieurs termes, qu'il est possible de répartir en trois catégories dont l'équilibrage constitue en pratique l'une des difficultés principales du réglage de l'entraînement. Un premier terme lié à tâche à proprement parler (atteinte de l'objectif), des termes de sécurité (retournement du robot, orientation), et des termes de qualité de mouvement (progression, marche arrière, cap, temps). L'ordre de ces tâches correspond à leur importance, par exemple, une pénalité de sécurité mal réglée peut annuler tout apprentissage de la tâche principale, c'est pourquoi leur réglage progressif est important. Plus de détail sur la création des récompense et leur ajustement sont détaillés en section 5.2. L'entraînement de cette architecture de référence est initialement mené avec des objectifs situés à proximité du robot, sur un terrain plat sans obstacle, afin de garantir la convergence rapide de l'ensemble de ces termes avant d'aborder le curriculum de distance présenté en section suivante.

Curriculum d'apprentissage sur la distance à l'objectif

Une fois la convergence obtenue sur des objectifs proches, un curriculum d'apprentissage est mis en place afin d'étendre progressivement la portée de la politique apprise. Le principe consiste à augmenter graduellement, au fil de l'entraînement la distance entre la position initiale du robot et son objectif. Cette approche progressive limite le risque que l'agent reste bloqué dans un optimum local associé à une politique d'exploration insuffisante, qui serait possible avec des objectifs lointains en début d'apprentissage provoquant de faibles récompenses. Le seuil de distance déclenchant le passage à l'étape suivante du curriculum est conditionnée par le taux de réussite observé sur les environnement parallèles. Lorsqu'un certain nombre d'épisodes sont passées et que ce taux dépasse un certain seuil alors la distance d'apparition de l'objectif change et le cycle est répété.

Généralisation aux terrains à friction variable

La deuxième étape de compléxification du modèle consiste à introduire un terrain moins idéal, en introduisant le terrain généré par l'algorithme présenté en section 3.3 constitué de zone de frictions variées permettant ainsi de représenter un sol d'herbe, de gravier ou d'asphalte sous la forme de parterres aléatoires. De surcroît, les propriétés de friction associées à chaque roue sont variées indépendamment d'un robot à l'autre au sein d'un même lot d'entraînement. Cette *randomisation* de domaine vise à empêcher la politique de sur-apprendre un modèle de dérapage particulier, et à la contraindre à développer un comportement robuste face à l'incertitude sur les paramètres de contact. L'observation, l'action et l'architecture du réseau restent inchangées par rapport à l'architecture précédente, permettant d'évaluer la capacité de généralisation du modèle simple avant d'en augmenter la complexité architecturale.

Intégration de la carte de coût locale

La dernière étape de complexification vise à exploiter, en complément du vecteur d'observation précédent, l'information de la carte de coût produite par le robot, afin de permettre l'évitement d'obstacles directement au sein de la politique apprise.

Une première tentative, consistant à aplatir la carte de coût sous la forme d'un vecteur et à le concaténer directement au vecteur d'observation avant de la fournir à un unique MLP a été mise en oeuvre à titre de comparaison. Comme attendu, ce choix architectural est voué à l'échec puisque la mise à plat de la carte de coût détruit la structure spatiale bidimensionnelle de l'information. De plus la dimension du vecteur d'entrée obtenu après aplatissement est très largement supérieure à celle des sept variables d'observation initiales, ce qui déséquilibre l'influence relative de ces derniers dans les premières couches du réseau et complique un apprentissage conjoint. Le schéma 4.3

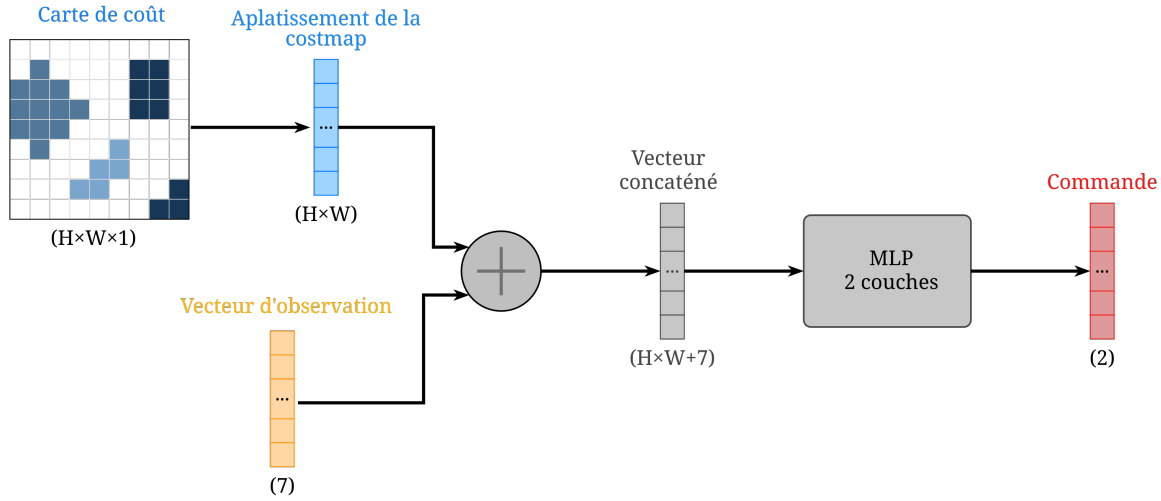


FIGURE 4.3 – Schéma simple d'une architecture de fusion naïve de la costmap

Les résultats détaillés de cette architecture et les autres essais faits avec celui-ci sont détaillés en section 5.2.

Encodeur convolutif de la carte de coût

Une stratégie plus judicieuse d'intégration de la carte de coût est, au contraire de l'architecture précédente, d'utiliser des opérateurs conservant les propriétés spatiales de l'information. Un encodeur convolutif indépendant du vecteur d'observation peut permettre le traitement de la carte de coût. Celui-ci est décomposé en deux couches de convolution, chacune suivie d'une opération de *pooling*, avant d'être aplatie et projetée par une couche *fully connected* sur un vecteur de caractéristiques de taille 64. Cette architecture

permet d'exploiter la structure spatiale de la carte de coût, les couches convolutives partagent leurs paramètres sur l'ensemble de l'image et sont ainsi en mesure de détecter des motifs locaux (bords d'obstacles, zones d'encombrement) indépendamment de leur position. Les couches de *pooling* permettent de réduire progressivement la résolution spatiale. Le vecteur de sortie de dimension 64 constitue ainsi un résumé compact de l'information contenue dans la carte de coût et est de dimension comparable à celle du vecteur d'observation principal limitant le déséquilibre de représentation.

Architectures de fusion des observations

Deux architectures de fusion entre la représentation issue de l'encodeur de carte de coût et le vecteur d'observation de base sont proposées et comparées.

Architecture à fusion précoce : Dans la première architecture, le vecteur de caractéristiques de taille 64 issu de l'encodeur convolutif est directement concaténé au vecteur d'observation de sept valeurs, avant d'être transmis à un unique MLP chargé de produire la commande finale, tel que le schéma 4.4 le présente.

Cette architecture simple laisse au réseau la responsabilité de mettre en relation, dès ses premières couches, les informations de bas niveau relatives à la carte de coût et celles, relatives à l'état cinématique du robot et à l'objectif.

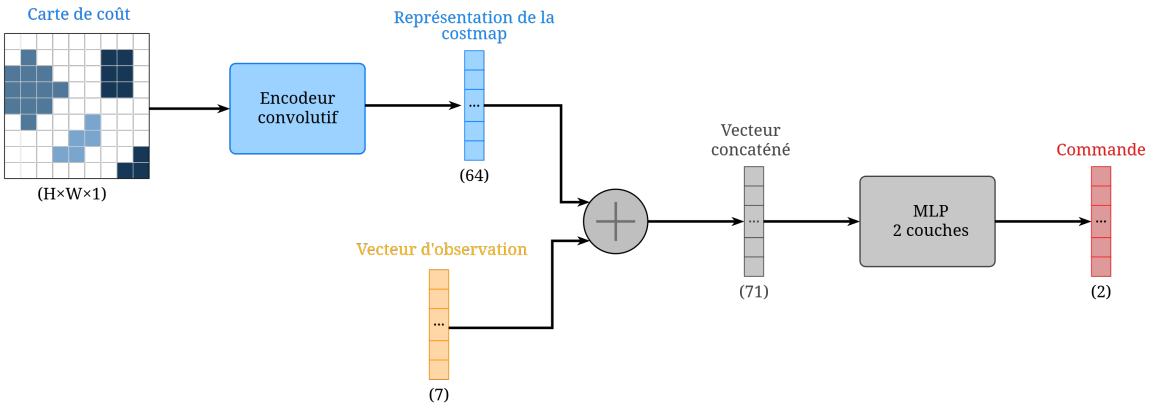


FIGURE 4.4 – Schéma simple de l'architecture de fusion précoce de la costmap

Architecture à fusion tardive. Dans la seconde architecture, le vecteur d'observation de sept valeurs est d'abord traité par son propre MLP, produisant une représentation intermédiaire de l'état du robot, avant que celle-ci ne soit concaténée à la sortie de l'encodeur de carte de coût. La représentation combinée est ensuite transmise à un second MLP, distinct du premier, qui produit la commande finale. Cette architecture est représentée en figure 4.5.

Ces deux architectures reposent sur des hypothèses différentes quant à la manière dont l'information géométrique de la carte de coût doit interagir avec l'état cinématique du robot.
L'architecture à fusion précoce permet au réseau d'apprendre des interactions non linéaires entre les deux sources d'information dès les premières couches, ce qui peut s'avérer bénéfique si la décision de commande dépend de combinaisons complexes entre la position de l'objectif et la configuration locale des obstacles.

L'architecture à fusion tardive, à l'inverse, impose une forme de traitement modulaire : le premier MLP peut être vu comme extrayant un résumé de l'état cinématique indépendamment de la perception de l'environnement, la fusion n'intervenant qu'au niveau d'une représentation déjà abstraite. Cette seconde approche est susceptible de faciliter l'apprentissage en réduisant l'interférence entre les deux sources d'information dans les couches précoces du réseau, au prix d'une capacité de représentation conjointe potentiellement plus limitée.

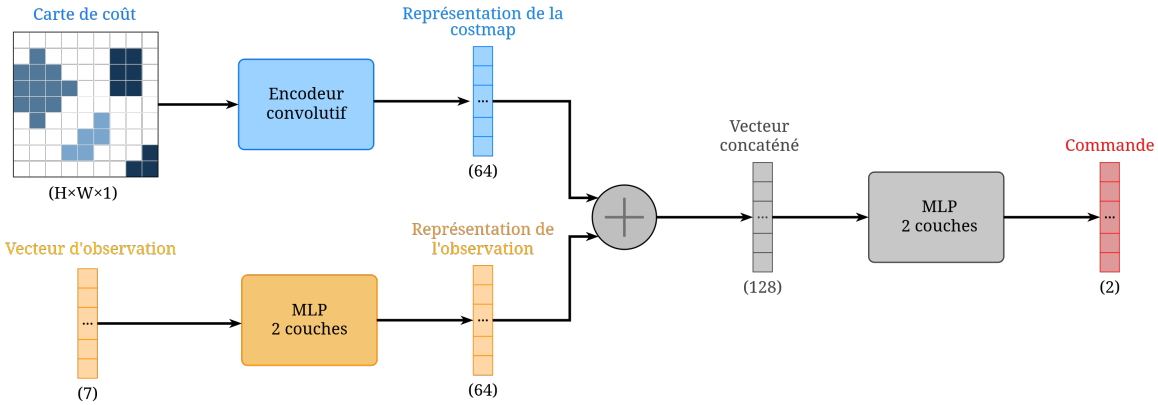


FIGURE 4.5 – Schéma simple de l'architecture de fusion tardive de la costmap

4.3 Implémentation dans Isaac Lab

Définition des tâches d'apprentissage

Chaque architecture présentée en section 4.2 est implémentée sous la forme d'une tâche (*task*) Isaac Lab, définie par une classe de configuration d'environnement regroupant l'ensemble des éléments nécessaires à l'entraînement. Définition de la scène (robot, terrain, obstacles éventuels), espace d'observation et d'action, fonction de récompense, conditions d'arrêt d'épisode (atteinte de l'objectif, collision, dépassement du temps imparti), ainsi que les paramètres d'apprentissage, sont tous écrits dans un fichier. Cette structure en tâches indépendantes permet de faire évoluer indépendamment chaque composante du problème, permettant la complexification progressive justifiée dans la section précédente. Le passage d'un terrain plat à un terrain à friction variable, ou l'ajout de la carte de coût, se traduit ainsi par la définition d'une nouvelle tâche héritant de la précédente où ne sont modifiés que les éléments pertinents (typiquement le poids des récompenses).

Principe des environnements parallèles

Isaac Lab s'appuie sur le simulateur d'Isaac Sim, présenté au chapite 3 , pour instancier simultanément plusieurs centaines de copies indépendantes du robot sur un même Graphical Processing Unit (GPU). Chaque instance dispose de son propre robot, de son propre objectif et de ses propres paramètres de friction. Mais toutes sont simulées en parallèle au sein d'un unique monde physique dans lequel les robots peuvent entrer en collisions avec les obstacles mais pas entre eux puisque provenant d'*environnements* différents. À chaque pas de temps, l'ensemble des observations issues des différentes instances est regroupé en un lot unique (*batch*) transmis à la politique, qui produit en retour un lot de commandes appliquées simultanément à chaque instance. Ce fonctionnement vectorisé, très différent d'une simulation classique exécutée en temps réel sur Central Processing Unit (CPU) est ce qui permet de générer en quelques secondes un volume de données équivalent à plusieurs heures d'interactions séquentielles. Ce qui explique l'intérêt d'Isaac Lab pour un algorithme *on-policy* tel PPO et la possibilité pratique de mettre en oeuvre un curriculum d'apprentissage réactif, le taux de réussite sur l'ensemble des environnements étant évalué à haute fréquence. De plus, Isaac sim permet la création de vues de débogage offrant la possibilité de voir l'entraînement se déroulant en temps réel 4.6. Cet affichage consomme cependant beaucoup de ressources de calcul et ralentit considérablement l'entraînement et n'est en pratique pas utilisé, l'outil *Tensorboard* lui est préféré.

Suivi de l'entraînement avec TensorBoard

Le suivi de la progression de l'apprentissage est assuré au moyen de *TensorBoard*, auquel le logiciel d'entraînement fourni périodiquement un ensemble de métriques agrégées sur l'ensemble des environnements parallèles. Les grandeurs suivies incluent notamment la récompense moyenne par épisode, la longueur moyenne des épisodes, le taux de réussite, ainsi que des indicateurs internes à l'algorithme PPO, tels que la

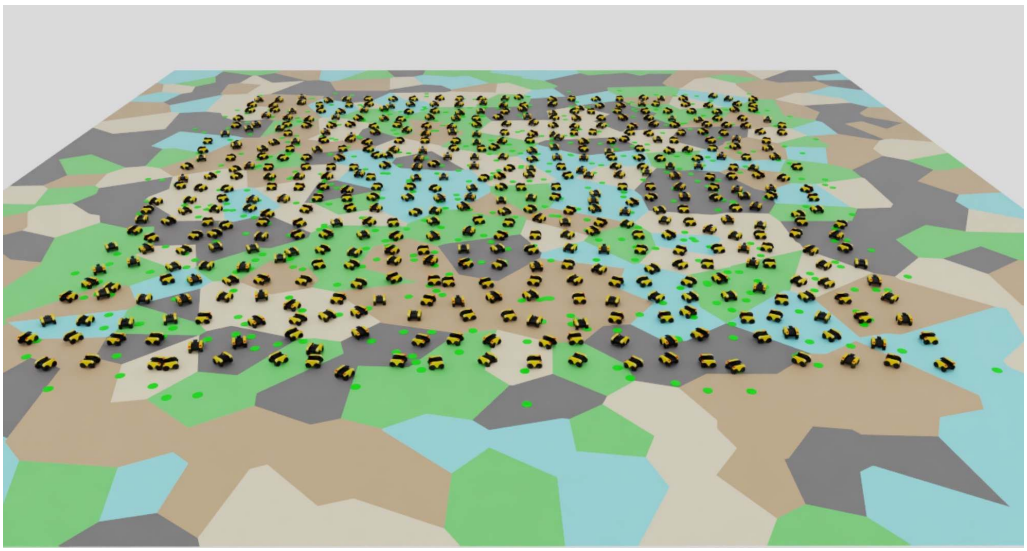


FIGURE 4.6 – vue de débogage permettant d'observer les robots pendant leur entraînement

valeur de la fonction de perte (*loss*), l'entropie de la politique. Le suivi conjoint de ces différentes courbes permet de diagnostiquer déventuelles instabilités au cours de lentraînement, telles quune diminution brutale de lentropie, pouvant indiquer une convergence prématurée de la politique vers un comportement déterministe, une stagnation du taux de réussite ou encore une dégradation de la récompense cumulée. Lanalyse de ces indicateurs permet ainsi dorienter lajustement des hyperparamètres et, plus généralement, dévaluer la qualité de lapprentissage. Actualisées en temps réel pendant lentraînement, ces courbes permettent également de déterminer si un arrêt anticipé est nécessaire lorsquune dérive des métriques dapprentissage met en évidence un paramétrage inadéquat, notamment au niveau des hyperparamètres ou des poids des différentes récompenses. Certains entraînements pouvant durer plusieurs heures, la détection précoce de ces comportements permet déviter de mobiliser inutilement des ressources de calcul et de relancer rapidement un entraînement avec une configuration mieux adaptée.

Chapitre 5

Expérimentations et résultats

Ce chapitre présente les expérimentations réalisées afin d'évaluer les différentes stratégies de commandes étudiées au cours de ce travail. Des essais ont été menés à la fois en conditions réelles, sur un robot *Agilex Scout2.0*, et en simulation sur un robot *Clearpath Warthog*, dans l'environnement Isaac Sim et Isaac Lab. Les expérimentations permettent ainsi de mesurer successivement les performances de contrôleurs classiques proposés par `Nav2`, en particulier le DWB et le MPC, puis des approches fondées sur l'apprentissage par renforcement.

L'objectif n'est pas uniquement de vérifier le fonctionnement de ces différentes approches, mais également d'identifier leurs limites pratiques et les difficultés associées à leur paramétrage.

5.1 Tests réels

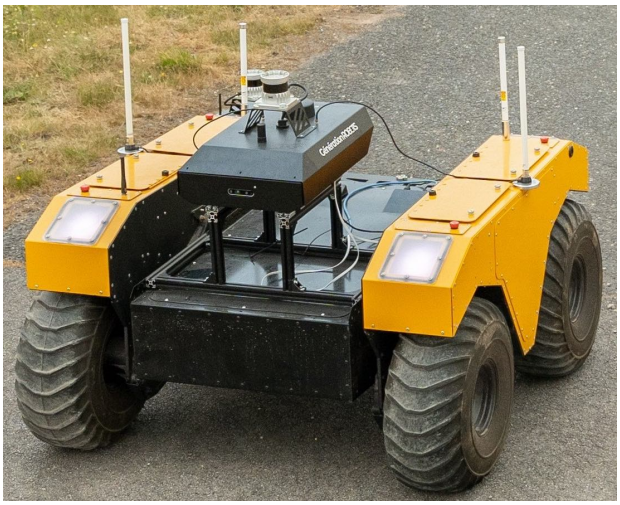
Plateformes expérimentales et intégration matérielle

Les expérimentations en conditions en réelles ont été réalisées à l'aide de deux plateformes robotique tout-terrain : le *CLearpath Warthog* et l'*Agilex Scout2.0*. Ces deux robots sont des plateformes à quatre roues motrices et directrices adaptées aux déplacements sur des terrains destructurés. Leurs principales caractéristiques sont résumées dans le tableau 5.1.

Caractéristique	**Warthog**	**SCOUT2.0**
Dimensions	$1.52 \times 1.38 \times 0.83$ m	$0.930 \times 0.699 \times 0.349$ m
Masse	280 kg	68 kg
Charge utile	272 kg	50 kg
Vitesse limite	5.0 m/s	1.5 m/s

TABLE 5.1 – Caractéristiques principales des robots utilisés.

Les deux plateformes sont intégrées à une architecture logicielle reposant sur ROS 2 Jazzy. Afin de centraliser la perception et l'intelligence embarquée, chaque robot est équipé d'un module d'extension (*ROSKit*) développé par *Génération Robots*. Ce kit regroupe une unité de calcul mixte composée d'un mini-PC (*Asus NUC 15 Pro* avec 16 Gb de mémoire vive sur le Scout2.0 et 64 Gb sur le Warthog) et d'une carte d'accélération matérielle (*NVIDIA Jetson AGX Orin*), couplée à une suite de capteur : une centrale inertielle (*Phidget Spatial Precision 3/3/3*), un Light Detection And Ranging (LiDAR) 3D (*Ouster OS1 rev7* 32 nappes pour le Scout2.0 et 128 nappes pour le Warthog), un récepteur GNSS (*RTK2B Simple*) et une caméra stéréoscopique (*Stereolabs ZED X*). L'ensemble des algorithmes de traitement, de perception et de contrôle sont exécutés localement sur ce kit. La communication entre le *ROSKit* et la base roulante du robot est assurée par un bus CAN pour le Scout2.0. Ce bus permet la transmission descendante des consignes de vitesse et la remontée de l'état de la plateforme (état de charge des batteries, déclenchement des sécurités, allumage des phares, et encodeurs des roues). Les deux robots ains que leur ROSKit sont représentés en figure 5.1



(a) Clearpath Warthog utilisé pour les expérimentations réelles

(b) Agilex Scout2.0 utilisé pour les expérimentations réelles

FIGURE 5.1 – Les deux plateformes robotiques étudiées, le boîtier noir sur le dessus contient l'ensemble des capteurs et les ordinateurs de bord.

Conditions expérimentales

Bien que les deux plateformes soient disponibles pour les expérimentations, les essais en conditions réelles ont été réalisés sur le Scout2.0. Le Warthog, de par son gabarit et son poids important, présentait des risques matériels trop élevés pour des phases de mise au point initiales d'algorithmes de conduite agressive.

Les campagnes d'essais réels se sont déroulées sur deux semaines consécutives sur le site d'essais du projet Pendragon. Ce terrain est caractérisé par des zones urbanisées reproduisant un village avec routes et bâtiments, et par des zones plus destructurées de routes de gravier et de prairies accidentées.

La première semaine de campgne a été consacrée à la prise en main du *ROSKit*, et des bases de code existantes sur le robot. Étape nécessaire afin de maîtriser la chaîne complète permettant d'envoyer des commandes au robot et de récupérer les données des capteurs.
La seconde semaine a permis de tester et de caractériser les contrôleurs locaux DWB et MPC de `Nav2`. L'objectif cible était l'obtention d'un comportement de conduite agressive.

Déploiement des contrôleurs classiques

Le déploiement des contrôleurs classiques en conditions réelles a permis de mettre en évidence plusieurs limites techniques majeures, relevant à la fois de la chaîne de perception et de localisation, de l'architecture logicielle de `Nav2`, et du réglage des contrôleurs eux-mêmes

Concernant la chaîne de navigation, des défaillances ont été observées, se traduisant par une dérive entre les odométries locale et globale, dégradant ainsi le positionnement relatif du robot. Bien qu'a priori indépendant de la problématique de contrôle, ce phénomène s'est révélé pénalisant dans la mesure où `Nav2` requiert une estimation précise de la position du robot pour fonctionner correctement, celle-ci étant utilisée à chaque instant pour exprimer la position de l'objectif dans le repère local du robot. Aucune solution simple n'a été identifiée pour fournir uniquement une position relative de l'objectif, même dégradée, sans recourir à une localisation complète du robot. Cette difficulté est renforcée par le fort couplage entre les briques fonctionnelles de `Nav2`, qui rend difficile leur isolement en vue de tester une fonctionnalité unique.

Par ailleurs, même après désactivation temporaire des briques d'évitement d'obstacles au sein des algorithmes DWB et MPC, réalisée afin d'évaluer la vitesse maximale atteignable par ces contrôleurs, des limitations de vitesse persistantes ont continué à être appliquées, empêchant le robot d'atteindre ses performances limites.

Le paramétrage des contrôleurs s'est en outre avéré particulièrement complexe et largement empirique.

La structure modulaire de `Nav2` permet certes de disposer rapidement d'une chaîne de navigation fonctionnelle, mais limite la possibilité de modifier facilement en profondeur son architecture ou d'y intégrer des fonctionnalités développées sur mesure.

Malgré ces contraintes, le robot est parvenu de manière répétable à atteindre la pose finale demandée tout en contournant les obstacles rencontrés sur son parcours. Les tests réalisés comprennent notamment une navigation entre deux points d'une route ainsi qu'un contournement de bâtiment. La réussite de ces essais valide le fonctionnement principal de la navigation classique en environnement réel.

Ces expérimentations ont ainsi permis de valider l'intégration matérielle et logicielle ainsi que le fonctionnement global de la chaîne de navigation. Elles ont également mis en évidence les difficultés pratiques liées au réglage de contrôleurs génériques, ce qui a motivé la poursuite des expérimentations en simulation, où les conditions expérimentales peuvent être contrôlées de manière beaucoup plus fine.

5.2 Tests simulés

Les expérimentations en simulation visent à évaluer les algorithmes développés dans un environnement virtuel contrôlé, progressivement élaboré sous NVIDIA Isaac Sim et Isaac Lab. Ces essais ont été menés selon deux axes. Le premier concerne la validation de la construction d'une simulation compatible avec ROS 2. Le second concerne l'entraînement et l'évaluation de contrôleurs fondés sur le RL.

Validation de la chaîne de simulation avec ROS 2

La première phase d'essais en simulation consiste à valider la simulation construite en section 3.1. On cherche d'abord à valider la communication entre l'environnement Isaac Sim et l'écosystème ROS 2 via le module *ROS Bridge* du logiciel. La disponibilité et l'intégrité des flux de données capteurs ont été vérifiées sous RViz (logiciel de visualisation de ROS 2 développé à des fins de débogage). Un asservissement manuel basique à l'aide du noeud `mouse_teleop` a permis de valider la réponse dynamique du modèle virtuel du robot aux commandes envoyées. Ensuite, une architecture conteneurisée basée sur Docker (exécutant Ubuntu 24.04 et ROS 2 Jazzy) a été déployée. La pile `Nav2` y a été instanciée avec le contrôleur DWB, alimentée directement par des données de localisation parfaites (via un noeud intermédiaire, générant le *TF*) et par la donnée LiDAR brute. La figure 3.2 illustre le robot dans cette phase d'essai. Cette configuration volontairement simplifiée, affranchie des incertitudes liées à la fusion de capteurs, a servi de preuve de concept. Les ordres de mission transmis via RViz ont confirmé la capacité du robot simulé à réagir de manière cohérente aux consignes. Confirmant le fonctionnement de tous les éléments de la simulation Isaac Sim.

Apprentissage par renforcement, modèle de base et stabilisations

La seconde phase d'essais en simulation a permis d'entreprendre l'entraînement d'agents par RL. La première itération visait à valider le cycle d'apprentissage sur un environnement plat et épuré. La fonction de récompense initiale R_{init} ne prenait en compte que l'atteinte du point objectif, le taux de progression vers l'objectif (rapprochement), l'alignement du cap du robot avec l'objectif. et un dernier terme de pénalité constante à chaque pas de temps pour contraindre le robot à agir. Un robot est considéré comme ayant atteint son obje

$$R_{\text{init}} = w_{\text{obj}} \cdot \mathbb{1}_{d_t < d_{\text{seuil}}} + w_{\text{progres}} \cdot (d_{t-1} - d_t) + w_{\text{cap}} \cdot \cos\theta_o bj + w_{\text{temps}} \tag{5.1}$$

Où les w_i représentent les poids de chaque récompense, d_{seuil} la distance seuil pour atteindre l'objectif, d_{t-1} et d_t les distance à l'objectif au pas de temps précédent et actuel et w_{cap} l'angle entre le cap du robot et l'objectif. La fonction de récompense est calculée tous les trois temps de simulation afin d'obtenir un contrôle à 20 Hz, pendant toute la durée d'un épisode de quinze secondes. Passée ce temps, l'épisode est arrêté et un nouveau est lancé si le robot n'a pas déjà réussi à atteindre son objectif. Ainsi un épisode ne s'interrompt que si le robot atteint son objectif ou s'il n'arrive pas à le faire pendant le temps imparti. L'objectif est

considéré comme atteint dès que le centre du robot est à une distance inférieure à d_{seuil}. Cette formulation simple permet de définir les comportements fondamentaux attendus sans introduire un nombre excessif de contraintes dans l'apprentissage.

Les premiers résultats issus d'un entraînement avec 1024 robots parallèles, ont révélé un comportement indésirable ; l'agent appliquait des accélérations longitudinales trop fortes, provoquant le basculement du robot. De plus le robot pouvait se déplacer en marche arrière. Bien que ce dernier comportement ne soit pas nécessairement superflu, la marche avant reste à privilégier pour que le robot développe un comportement intuitif.
Pour pallier ces phénomènes, trois pénalisations ont été ajoutées. Premièrement une pénalité sur l'attitude (en roulis et tangage) incitant le robot à maintenir une assiette plane. Deuxièmement l'introduction d'une condition d'arrêt anticipée de l'épisode associée à une forte pénalité en cas de retournement du véhicule. Enfin une pénalité sur les vitesses négatives afin d'inhiber la marche arrière systématique.

$$\begin{cases} e_{\text{orient}} = g_x^2 + g_y^2 \\ p_{\text{arriere}} = w_{\text{arriere}} \cdot [\mathbb{1}_{v_x<0} \cdot v_x]^2 \end{cases} \tag{5.2}$$

Avec g_x, g_y les composantes selon les axes $\hat{x}$ et $\hat{y}$ du vecteur de gravité dans le repère du robot, e_{orient}, l'erreur d'assiette, v_x la vitesse d'avance du robot et p_{arriere} la pénalité sur la marche arrière appliquée. L'erreur d'assiette est constamment appliqué avec un poids w_{orient} et est réutilisée pour déterminer quand le robot s'est retourné en la comparant à un valeur seuil $e_{\text{orient, seuil}}$.

À l'issue de ces modifications, le modèle de base sur terrain plat a montré une convergence satisfaisante, permettant au robot d'atteindre sa cible avec une trajectoire fluide. La figure 5.2 montre les robots dans l'environnement de base en plein entraînement, les disques vers sont les objectifs à atteindre avec leur rayon représentant d_{seuil}.

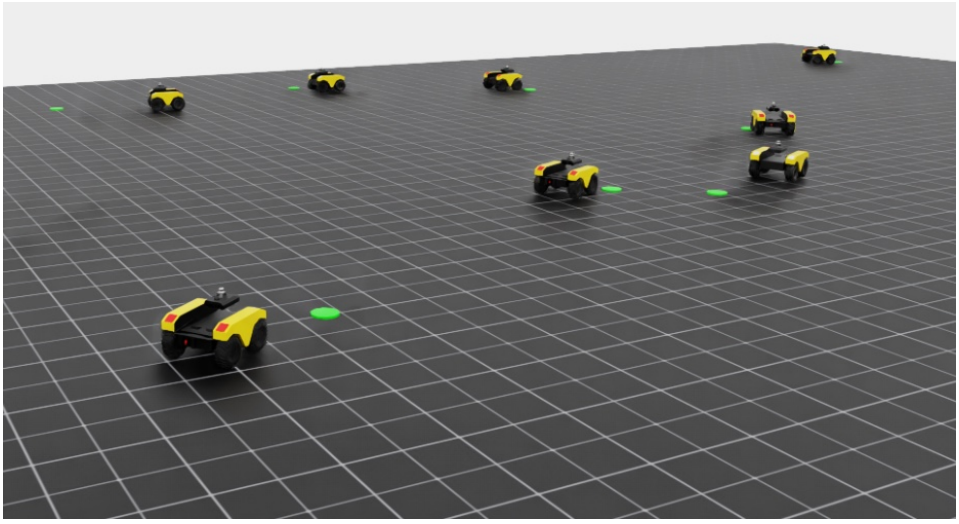


FIGURE 5.2 – Les robots dans leur environnement de base

Curriculum sur la distance à l'objectif et régularisation de la commande

À la suite des premiers résultats encourageants, une nouvelle tâche d'apprentissage a été développée, introduisant un curriculum portant sur la distance initiale séparant le robot de l'objectif. Les niveaux successifs éloignent progressivement la cible, augmentant ainsi la difficulté.

L'agent définit de la même manière qu'à la section précédente parvient à atteindre l'objectif sur l'ensemble des niveaux, mais l'analyse des comportements obtenus sur les derniers d'entre eux révèle une stratégie indésirable. Le robot accumule une vitesse excessive à l'approche de l'objectif et le manque régulièrement, poursuivant sa trajectoire sans parvenir à s'arrêter. Pour corriger ce comportement deux termes de pénalité ont été ajoutés à la fonction de récompense : une régularisation des commandes et une limitation de la

vitesse en fonction de la distance à l'objectif. Soit $a_t = [v_{cmd,t}, \omega^T_{cmd,t}]$ le vecteur d'action à l'instant t et $\Delta v_t = v_{cmd,t} - v_{cmd,t-1}$ l'écart de commande linéaire entre deux instant successifs. La régularisation des commandes s'écrit :

$$P_{\text{acc}} = \text{ReLU}(\Delta v_t), \qquad P_{\text{dec}} = \text{ReLU}(-\Delta v_t), \qquad P_{\text{ang}} = (\omega_{cmd,t} - \omega_{cmd,t-1})^2 \tag{5.3}$$

Les termes P_{acc} et P_{dec} sont volontairement dissymétriques afin de pénaliser plus sévèrement les accélérations brutales tout en autorisant une décélération franche à l'approche de la cible. P_{ang} limite quant à elle les variations abruptes de la commande en rotation.

Une vitesse cible v_{target}, décroissante avec la distance d à l'objectif, est par ailleurs définie par une tangente hyperbolique, et une pénalité P_{vitesse} sanctionne son dépassement par la vitesse réelle v_x du robot dans son repère local.

$$v_{\text{target}}(d) = v_{\max} \cdot \tanh\left(\frac{d}{2}\right), \qquad P_{\text{vitesse}}(v_x, d) = \left(\text{ReLU}\left(v_x - v_{\text{target}}(d)\right)\right)^2 \tag{5.4}$$

Cette formulation assure une vitesse cible proche de $v_{\max}$ lorsque le robot est éloigné de l'objectif, et décroissante à son approche, incitant ainsi le modèle à ralentir avant d'atteindre la cible.

L'introduction de ces pénalités a permis de corriger le comportement de l'agent sur les niveaux avancés du curriculum ; le robot modère désormais son allure à l'approche de la cible et atteint celle-ci plus régulièrement, avec un taux de réussite supérieur. Cette amélioration est illustré par la courbe des récompenses enregistrée sous TensorBoard 5.3, où sont comparés l'apprentissage avec les récompenses de base (courbe bleu), l'ajout du curriculum sans les nouvelles récompenses (courbe orange) et avec curriculum et régularisations (courbe verte).

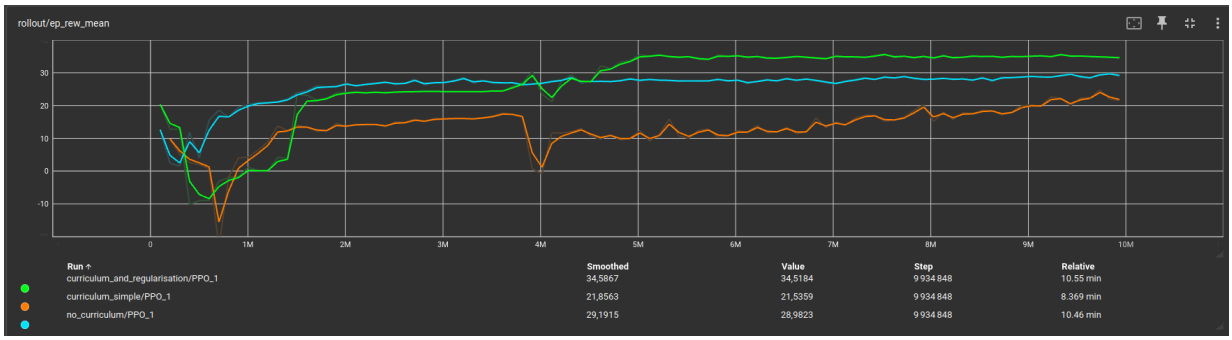


FIGURE 5.3 – Évolution de la récompense moyenne cumulée au cours de l'apprentissage, avec le système de récompense de base (bleu) , le curriculum sans régularisation (orange) et le curriculum avec régularisation (vert)

Découplage des dynamiques de rotation et de translation

Un second comportement suboptimal a été relevé ; le robot cherche systématiquement à s'aligner parfaitement avec la direction de l'objectif avant de démarrer son déplacement en translation. Si cette stratégie réduit l'erreur d'orientation, elle est encouragée par la formulation actuelle des récompenses et entraîne de ce fait une perte de temps inutile lorsque la cible est éloignée. La stratégie ne correspond pas non plus au comportement recherché pour un robot tout-terrain, pour lequel on préférerait qu'il commence à avancer tout en corrigeant progressivement son orientation.

Afin de découpler ces deux dynamiques, une pondération w_{heading} au terme d'alignement a été rajouté. Elle dépend de la distance d à l'objectif.

$$w_{\text{heading}}(d) = w_{\min} + (1 - w_{\min}) \cdot \left(1 - \tanh\left(\frac{d}{d_0}\right)\right) \tag{5.5}$$

avec $w_{\min} = 0{,}05$ et $d_0 = 5\,\text{m}$. Lorsque le robot est éloigné de l'objectif, w_{heading} tend vers sa valeur minimale, laissant la priorité à la translation, puis à mesure que la distance diminue, la pondération augmente et l'alignement devient alors progressivement prépondérant, conduisant le robot à s'orienter avec l'objectif. Cette formulation évite ainsi d'imposer une séparation stricte entre rotation et translaion, le robot pouvant progresser vers l'objectif tout en corrigeant son orientation en continu.

Intégration de la carte de coût

Les entraînements précédents ayant été validés sur un terrain plat comportant la segmentation de Voronoi pour le sol (voir la section 3.3), l'étape suivante a consisté à enrichir le vecteur d'observation de l'agent avec la carte de coût (section 3.4), afin de lui permettre d'exploiter explicitement les informations locales de traversabilité et de présence d'obstacles. L'utilisation de cette carte de coût devrait permettre de comparer les contrôleurs classiques et ceux issus d'un apprentissage automatique. En effet la plupart des contrôleurs classiques dont le DWB et le MPC repose sur une carte de coût générée à partir des capteurs par `Nav2`. Une première approche naïve a consisté à aplatir (*flatten*) la grille 2D de la carte de coût pour la concaténer directement au vecteur d'observation utilisé jusqu'ici. Conformément à nos attentes théoriques, cette tentative n'a pas permis à l'agent de converger vers une politique satisfaisante. La forte dimension de la représentation obtenue et l'absence de traitement spatial dédié empêchent le MLP d'exploiter la structure spatiale de la carte, dans laquelle les relations entre cellules voisines sont pourtant porteuses d'information.

Une expérience complémentaire a consisté à conserver cette architecture d'observation, mais en forçant à zéro l'ensemble des valeurs de la carte de coût avant concaténation. Une amélioration immédiate des performance de l'agent a été mesurée par rapport au cas précédent. Ce résultat, bien que contre-intuitif, confirme que l'injection brute de la carte de coût non prétraitée agissait principalement comme un bruit perturbateur dans le processus de décision de l'agent plutôt que commme une information exploitable, soulignant l'importance d'un prétraitement et d'une architecture de fusion adaptés à cette modalité d'observation. La figure 5.4, illustre d'ailleurs, à hyperparamètres et récompenses fixées, les variations de récompense reçues au cours des épisodes pour trois entraînements différents. En bleu la stratégie de la section précédente sans concaténation de la carte de coût, en orange la courbe d'entraînement avec concaténation de la carte de coût et enfin en vert la courbe où les valeurs concaténées sont mises à zéro.

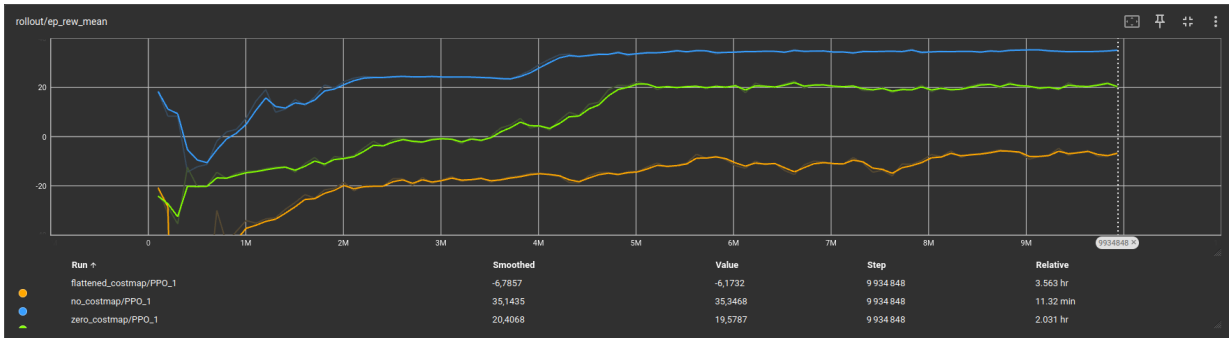


FIGURE 5.4 – Évolution de la récompense moyenne cumulée au cours de l'apprentissage, pour un modèle de base (bleu), avec concaténation de la costmap (orange), concaténation d'une costmap avec ses valeurs à zero (vert)

Architectures de fusions avancées

Les architectures de fusion plus élaborées présentées en fin de chapitre 4 visaient à répondre à la problématique de la représentation spatiale de la carte de coût en permettant un traitement spécifique de celle-ci avant sa fusion avec les observations vectorielles du robot. La définition formelle de ces architectures ainsi que l'adaptation de la tâche d'apprentissage à leurs différentes entrées ont été largement avancées, mais les contraintes temporelles n'ont pas permis de mener des sessions d'entraînement complètes jusqu'à convergence.

En l'absence de résultats quantitatifs, la pertinence de ces architectures ne peut donc être établie à ce stade et pourrait faire l'objet de travaux futurs. Leur conception constitue néanmoins une piste d'amélioration directe des expérimentations précédentes.

5.3 Analyse des résultats et discussion

L'ensemble des expérimentations menées, tant en conditions réelles qu'en simulation, met en évidence un point commun aux différentes approches étudiées : l'obtention d'un comportement satisfaisant dépend fortement des choix d'architecture et du réglage des paramètres, quelle que soit la méthode considérée. Plus précisément, le choix entre une approche classique (`Nav2` avec DWB/MPC) et une approche par RL repose sur un compromis entre effort de réglage paramétrique et effort de conception architecturale.

Contrôleurs classiques : Les expérimentations réelles ont montré qu'une chaîne de navigation générique permet effectivement au robot d'atteindre une pose cible tout en évitant les obstacles, avec des garanties formelles et une explicabilité du comportement supérieures à celles d'une politique apprise. L'obtention d'un comportement dynamique spécifique nécessite cependant un réglage important des paramètres, dont la complexité s'accroît lorsque l'environnement devient plus dynamique ou accidenté, et reste par ailleurs contrainte par la rigidité de l'architecture logicielle proposée par `Nav2`, qui complique l'intégration de contraintes sur mesure. Cette difficulté est accentuée par les problèmes de localisation et d'odométrie observés sur la plateforme réelle, qui rendent difficile la distinction entre les limitations propres au contrôleur et celles provenant des autres éléments de la chaîne de navigation.

Apprentissage par renforcement : Les approches par apprentissage par renforcement posent une problématique différente mais finalement comparable. Elles offrent une liberté nettement supérieure dans la définition du comportement recherché et permettent de faire émerger des stratégies de conduite complexes et adaptatives, la modification directe des observations, des récompenses, des pénalités ou de l'architecture du modèle étant possible à tout moment. Cette liberté déplace toutefois le problème vers la conception empirique de la fonction de récompense (*reward engineering*) et vers la structuration de l'espace d'observation. La pondération relative de chaque terme (progression vers l'objectif, attitude du robot, lissage des commandes) nécessite une expertise fine afin d'éviter la convergence vers des minima locaux ou l'émergence de comportements indésirables.

Les premiers entraînements illustrent bien ce couplage : la recherche d'une progression rapide vers l'objectif a d'abord conduit à des accélérations excessives et au renversement du robot, l'ajout d'une pénalité sur l'inclinaison a résolu ce problème tout en favorisant la marche arrière, une pénalité supplémentaire a limité ce nouveau comportement, avant que d'autres ajustements ne soient nécessaires pour contrôler vitesse et dynamique. Cette succession de corrections montre qu'une récompense ne contrôle jamais isolément le comportement pour lequel elle a été introduite : elle modifie l'équilibre global du problème d'optimisation et peut ainsi faire émerger des stratégies non anticipées. Le curriculum sur la distance à l'objectif fournit un second exemple de cette difficulté : l'augmentation progressive de la difficulté permet au modèle d'apprendre à atteindre des objectifs plus éloignés, mais fait apparaître sur les derniers niveaux une stratégie d'accélération excessive, que l'ajout d'une limitation de vitesse dépendant de la distance a permis de corriger, la pondération adaptative de l'alignement permettant ensuite d'ajuster la combinaison entre rotation et translation.

Ces résultats montrent que la conception d'une tâche de RL constitue en elle-même une problématique à part entière : le nombre de formulations possibles est très élevé, et le choix des coefficients associés aux différentes récompenses et pénalités repose largement sur une démarche empirique, nécessitant une compréhension fine du comportement observé pour identifier la composante responsable d'un défaut et la corriger sans dégrader les comportements déjà acquis. Les expérimentations menées dans ce chapitre ne permettent pas de désigner une approche comme universellement supérieure aux autres, mais elles dégagent un compromis entre flexibilité, complexité et maîtrise du comportement. Les contrôleurs classiques offrent

une solution rapidement intégrable et relativement robuste pour des tâches de navigation conventionnelles, mais leur architecture limite la recherche d'un comportement très spécifique. Les approches par RL offrent à l'inverse une plus grande liberté de conception, au prix d'une phase de développement et de réglage nettement plus lourde.

Chapitre 6

Conclusion

Ce travail visait à mettre en place un contrôleur robuste pour la navigation autonome d'un Unmanned Ground Vehicle (UGV) à direction par dérapage en milieu destructuré, en déclinant cet objectif en quatre axes : modélisation de la plateforme, simulation, analyse de contrôleurs classiques et développement d'un modèle d'apprentissage par renforcement pour le contrôle.

L'étude des modèles cinématiques et des contrôleurs DWB et MPC a mis en évidence une limite commune. La dépendance à une représentation explicite, plus ou moins fidèle, du comportement du robot et de son environnement, et la sensibilité au réglage empirique de nombreux paramètres.

La construction d'un jumeau numérique du Warthog sous Isaac Sim puis Isaac Lab, incluant l'automatisation de la conversion URDF vers USD, la génération procédurale de terrains segmentés et un capteur de carte de coût sémantique, a permis de disposer d'un environnement d'entraînement massivement parallélisable pour l'apprentissage par renforcement. L'entraînement d'agents PPO a permis de faire converger une politique de commande, du terrain plat jusqu'à la généralisation sur des sols à friction variable, par un travail itératif de conception de la fonction de récompense. Chaque terme ajouté fait varier l'équilibre global de l'optimisation, comme l'illustrent les corrections successives apportés pour éviter le retournement du robot ou les accélérations excessives. L'intégratin de la carte de coût par des architectures convolutives, en revanche, n'a pas pu être menée à convergence dans le temps imparti et constitue une perspective directe de poursuite.

En conclusion, la comparaison des approches étudiées ne dégage pas de méthode universellement supérieure, mais un compromis entre effort de réglage paramétrique et effort de conception architecturale. Les contrôleurs classiques offrent une solution rapidement intégrable mais rigide, tandis que le RL offre une liberté de conception supérieure au prix d'un *reward engineering* exigeant et sans garantie de convergence.

Bibliographie

[1] Clearpath Robotics. *Warthog UGV*. URL : https://clearpathrobotics.com/warthog-unmanned-ground-vehicle-robot.

[2] Agilex Robotics. *Scout V2.0*. URL : https://global.agilex.ai/products/scout-2-0.

[3] Turtlebot. *Turtlebot*. URL : https://www.turtlebot.com/.

[4] A. Segovla et M. Rombaut. « Continuous Curvature Path Finding for a Non-Holonomic Mobile Robot ». In : *IFAC Proceedings Volumes* 26.1 (1993). 1st IFAC International Workshop on Intelligent Autonomous Vehicles, Hampshire, UK, 18-21 April, p. 481-486. ISSN : 1474-6670. DOI : https://doi.org/10.1016/S1474-6670(17)49346-3.

[5] Dominic Baril et al. « Evaluation of Skid-Steering Kinematic Models for Subarctic Environments ». In : *CoRR* abs/2004.05131 (2020). URL : https://arxiv.org/abs/2004.05131.

[6] Dominic Baril et al. *DRIVE : Data-driven Robot Input Vector Exploration*. 2024. URL : https://arxiv.org/abs/2309.10718.

[7] Davide Locardi. « Modelling and control of a skid-steering mobile robot for indoor trajectory tracking applications ». Thèse de doct. Politecnico di Torino, 2020.

[8] Alvaro J Prado, Miguel Torres-Torriti et Fernando A Cheein. « Distributed tube-based nonlinear MPC for motion control of skid-steer robots with terra-mechanical constraints ». In : *IEEE Robotics and Automation Letters* 6.4 (2021), p. 8045-8052. DOI : https://doi.org/10.1109/LRA.2021.3102328.

[9] et al. S. Macenski. « From the desks of ROS maintainers : A survey of modern & capable mobile robotics algorithms in the robot operating system 2 ». In : *Robotics and Autonomous Systems* (2023).

[10] D. Fox, W. Burgard et S. Thrun. « The dynamic window approach to collision avoidance ». In : *IEEE Robotics & Automation Magazine* 4.1 (1997), p. 23-33. DOI : 10.1109/100.580977.

[11] Wubshet Ayalew et al. « Optimal path planning using bidirectional rapidly-exploring random tree star-dynamic window approach (BRRT*-DWA) with adaptive Monte Carlo localization (AMCL) for mobile robot ». In : *Engineering Research Express* 6 (2024). DOI : 10.1088/2631-8695/ad61bd.

[12] Open Navigation LLC and contributors. *DWB Controller*. Version Nav2. ROS 2 Navigation Framework. 2026. URL : https://github.com/ros-navigation/navigation2/tree/main/nav2_dwb_controller.

[13] Carlos E. García, David M. Prett et Manfred Morari. « Model predictive control : Theory and practiceA survey ». In : *Automatica* 25.3 (1989), p. 335-348. DOI : https://doi.org/10.1016/0005-1098(89)90002-2.

[14] Alvaro Paz et al. *Real-Time Nonlinear Model Predictive Control of Heavy-Duty Skid-Steered Mobile Platform for Trajectory Tracking Tasks*. 2025. URL : https://arxiv.org/abs/2510.02976.

[15] N. Koenig et A. Howard. « Design and use paradigms for Gazebo, an open-source multi-robot simulator ». In : 3 (2004), 2149-2154 vol.3. DOI : 10.1109/IROS.2004.1389727.

[16] NVIDIA Corporation. *NVIDIA Isaac Sim Documentation*. URL : https://docs.omniverse.nvidia.com/isaacsim/.

[17] Pixar Animation Studios. *OpenUSD : Universal Scene Description*. URL : https://openusd.org.

[18] Viktor Makoviychuk et al.. « Isaac Gym : High Performance GPU-Based Physics Simulation For Robot Learning ». In : (2021). URL : https://arxiv.org/abs/2108.10470.

[19] Mayank Mittal et al. « Isaac Lab : A GPU-Accelerated Simulation Framework for Multi-Modal Robot Learning ». In : *arXiv preprint arXiv :2511.04831* (2025). URL : https://arxiv.org/abs/2511.04831.

[20] Peter E. Hart, Nils J. Nilsson et Bertram Raphael. « A Formal Basis for the Heuristic Determination of Minimum Cost Paths ». In : *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), p. 100-107. DOI : 10.1109/TSSC.1968.300136.

[21] Ziqian Wu et Jiahao Liu. « Perlin noise and its improvements : A literature review ». In : *Applied and Computational Engineering* 77 (juill. 2024), p. 278-287. DOI : 10.54254/2755-2721/77/20240437.

[22] Allan Pinkus. « Approximation theory of the MLP model in neural networks ». In : *Acta Numerica* 8 (1999), p. 143-195. DOI : 10.1017/S0962492900002919.

[23] Wikimedia Commons. *Reinforcement learning diagram*. URL : https://commons.wikimedia.org/wiki/File:Reinforcement_learning_diagram.svg.

[24] John Schulman et al. « Proximal Policy Optimization Algorithms ». In : (2017). URL : https://arxiv.org/abs/1707.06347.

[25] Tuomas Haarnoja et al. *Soft Actor-Critic : Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. 2018. URL : https://arxiv.org/abs/1801.01290.

[26] Volodymyr Mnih et al. *Asynchronous Methods for Deep Reinforcement Learning*. 2016. URL : https://arxiv.org/abs/1602.01783.

Table des figures

Liste des tableaux

Annexes

Planification du stage

Caractéristiques des machines

Cette annexe recense les caractéristiques matérielles de la machine utilisée pendant le stage.

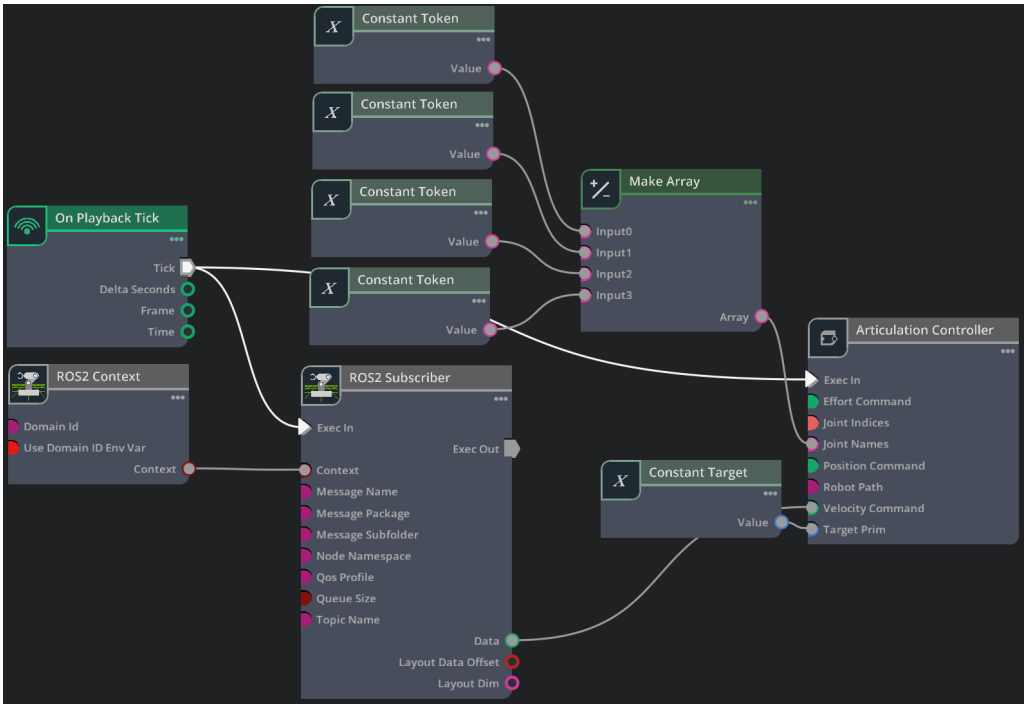
Composant	**Caractéristique**
CPU	Intel Core i7-14700HX 14 curs / 28 threads, 5,5 GHz max
RAM	32 Go DDR5
GPU	NVIDIA RTX 4000 Ada Generation Laptop GPU 12 Go VRAM Pilote 550.163.01
Stockage	NVMe Kioxia 3,7 To
Environnement	PyTorch 2.5.1 CUDA 12.1

TABLE .1 – Caractéristiques matérielles du PC de travail.

OmniGraph du Warthog sur Isaac Sim

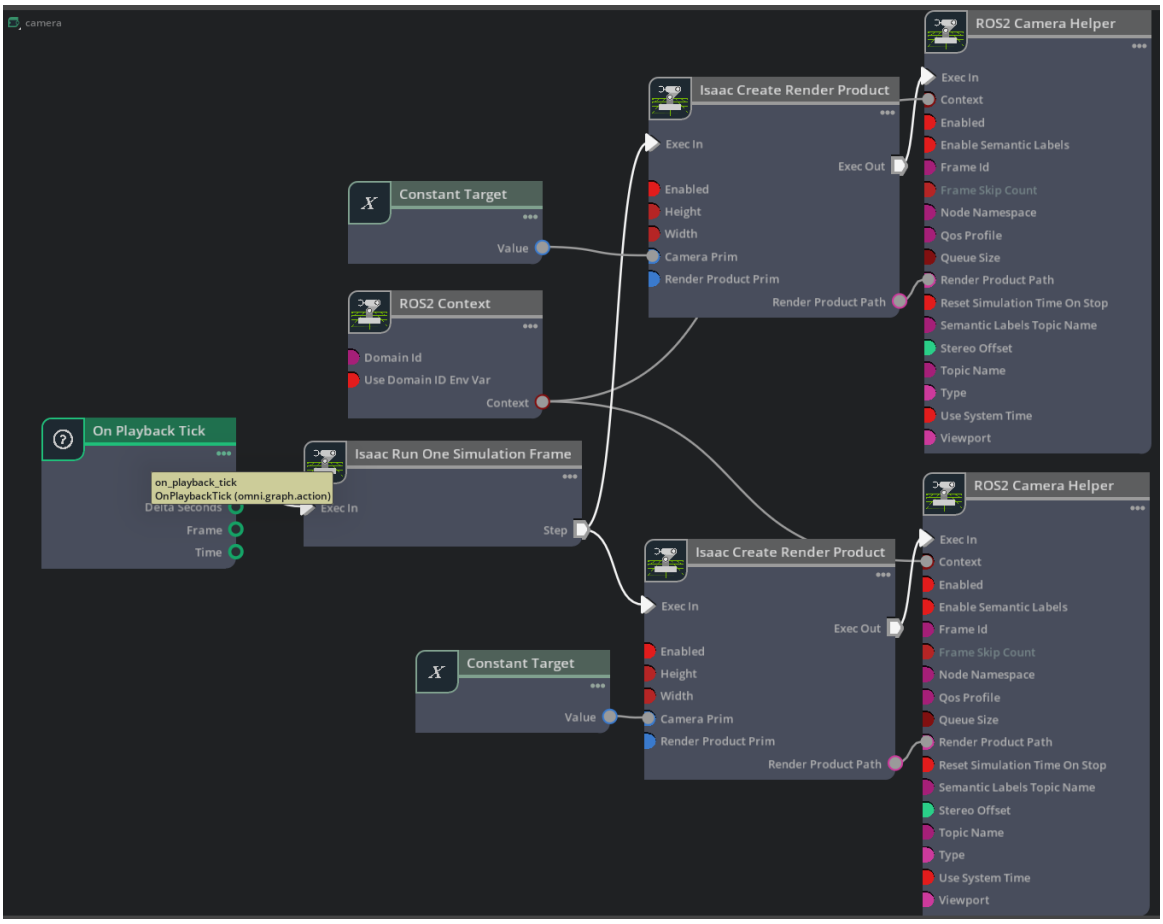
Cette annexe contient les omnigraph utilisés pour construire le *ROSBridge* entre Isaac Sim et l'extérieur de la simulation.

FIGURE .1 – Planification prévue et réelle du stage sous la forme d'un diagramme de Gantt

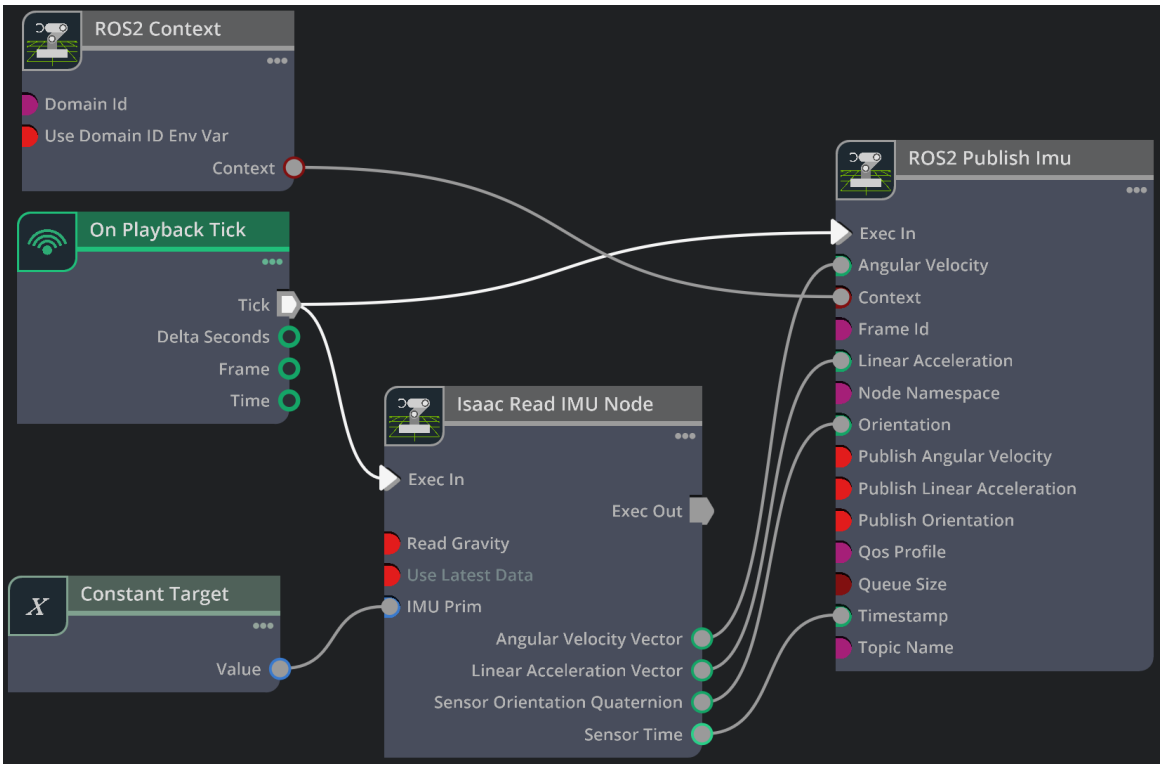


(a) Commande roue par roue

FIGURE .2 – Commande roue par roue du Warthog dans l'*OmniGraph*.

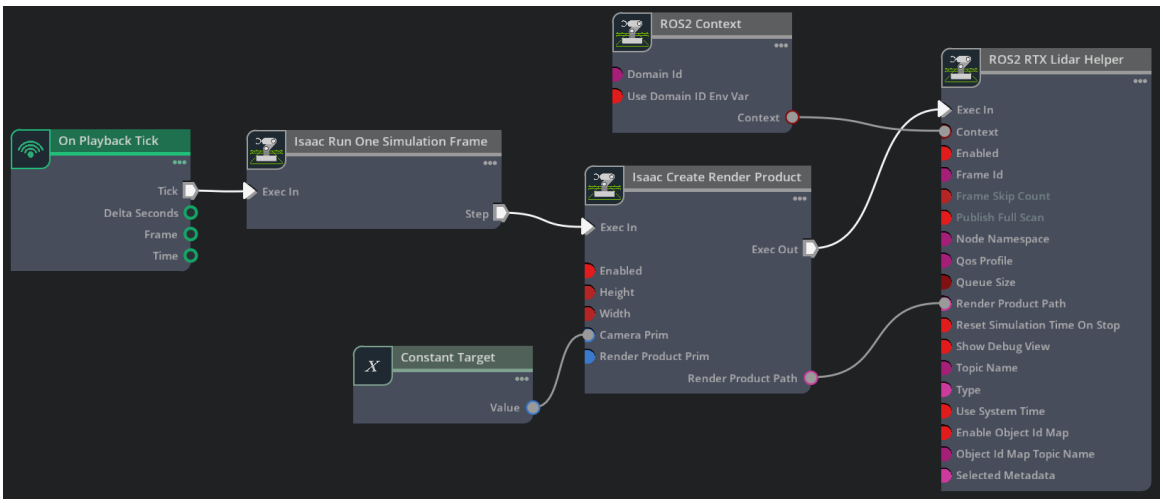


(a) Caméra stéréo

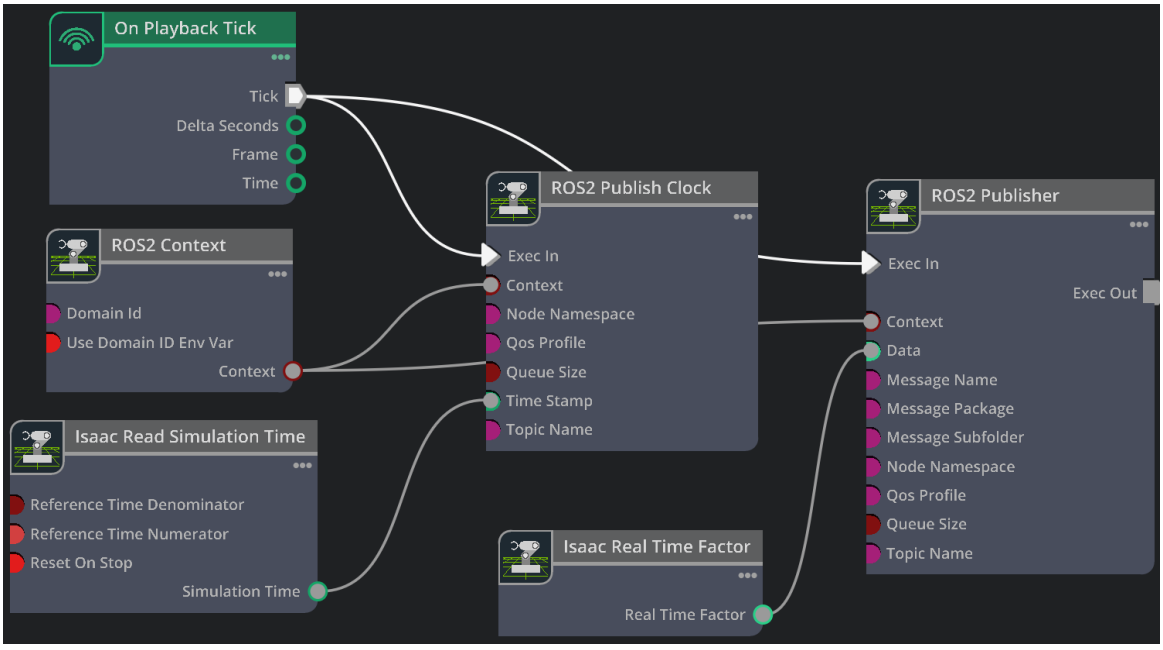


(b) Inertial Measurement Unit (IMU)

FIGURE .3 – Éléments de perception de l'*OmniGraph* du Warthog.

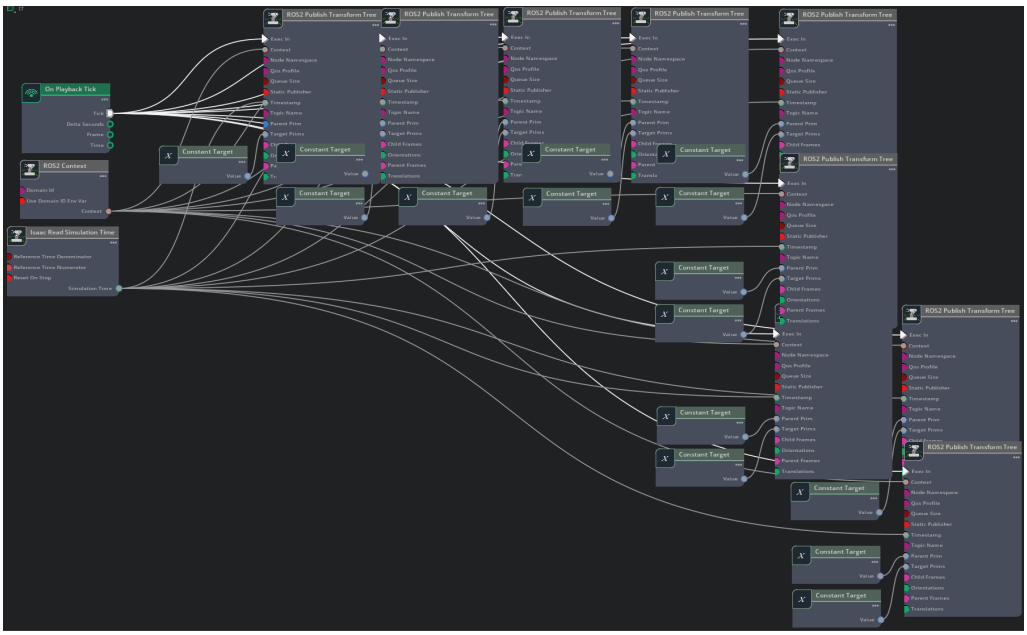


(a) LiDAR

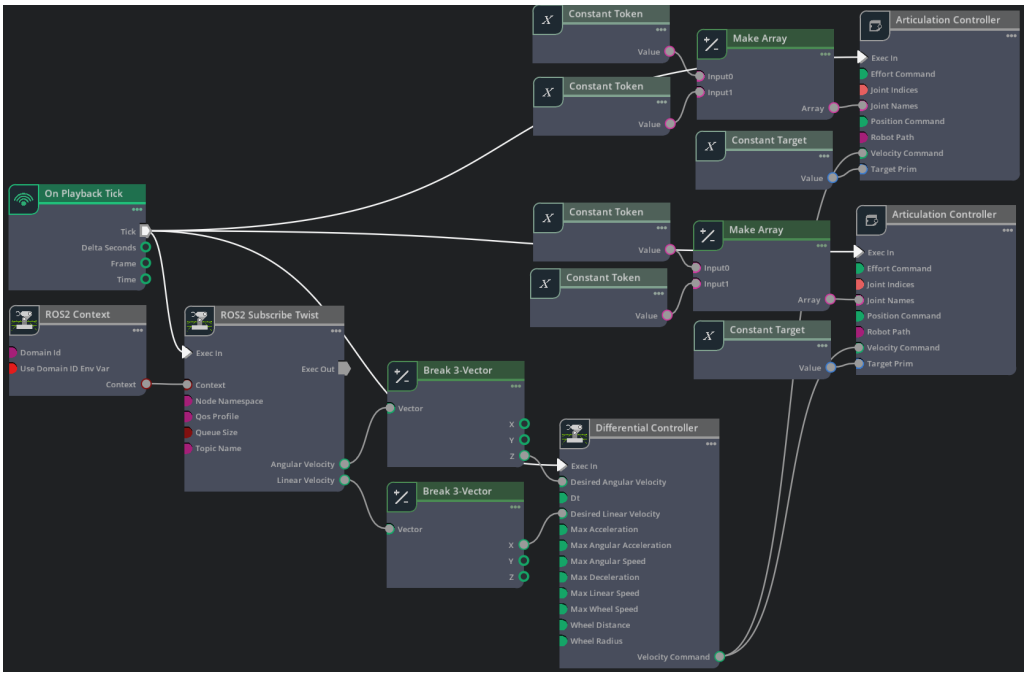


(b) Horloge et facteur temps réel

FIGURE .4 – Gestion des données temporelles et des capteurs dans l'*OmniGraph*.



(a) Arbre de transformations



(b) Commande par Twist

FIGURE .5 – Gestion des transformations et des commandes du Warthog dans l'*OmniGraph*.