RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

utbm
université de technologie
Belfort-Montbéliard

UNIVERSITÉ DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD

Exploitation d’un Large Vision Model (LVM) pour la navigation autonome d'un robot.

Mémoire TY22 - P2025

BALEZEAU Quentin
FISE-INFO

Université de Technologie de Belfort-Montbéliard
4 Rue Edouard Branly, 90000 Belfort
www.utbm.fr

Professeur encadrant
Dr. KAS Mohamed

Remerciement

Je tiens à exprimer ma profonde gratitude envers le Dr. KAS Mohamed pour son encadrement et sa patience tout au long de ce travail. Ses conseils avisés, sa disponibilité et son expertise ont été déterminants dans la réalisation de ce mémoire. Je lui suis particulièrement reconnaissant pour la confiance qu'il m'a accordée et pour m'avoir guidé avec bienveillance à chaque étape de ce projet.

Résumé

Ce rapport explore l'utilisation d'un Large Vision Model (LVM) pour la navigation autonome d'un robot, en se concentrant sur l'intégration de VLMaps dans un environnement simulé. L'objectif principal était de valider la faisabilité de l'utilisation de VLMaps pour générer des cartes sémantiques exploitables par un robot dans un environnement inconnu. Le document commence par une introduction au contexte général de l'utilisation des modèles de langage et de vision dans la robotique, soulignant les avancées récentes et les défis associés. Il présente ensuite un état de l'art des projets et publications récents dans le domaine, structuré autour de quatre axes principaux : raisonnement, planification, manipulation et navigation.

Le rapport détaille les différentes solutions explorées, notamment MoMa-LLM, ReKep, Navid et VLMaps, ainsi que les défis techniques rencontrés lors de leur installation et configuration. La méthodologie et l'installation sont décrites en détail, incluant la configuration de l'environnement, l'installation de ROS et des packages nécessaires, ainsi que le lancement des environnements de simulation. Bien que les scénarios de navigation n'aient pas été entièrement développés en raison de contraintes de temps, le rapport conclut avec un bilan personnel, soulignant les apprentissages tirés des défis rencontrés et les perspectives futures pour le projet.

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 Contexte général

Au cours des dernières années, de nombreuses équipes de recherche se sont mobilisées pour intégrer l'intelligence artificielle en robotique. Grâce à des techniques toujours plus avancées, nous assistons à l'émergence d'automates, de robots et de véhicules autonomes capables non seulement d'automatiser des processus industriels, mais aussi d'évoluer dans des environnements complexes et dynamiques pour nous aider au quotidien. Ces innovations ouvrent la voie à des systèmes polyvalents et intelligents, capables de comprendre et d'interagir avec leur environnement de manière de plus en plus naturelle.

Les robots ne se contentent plus d'exécuter des tâches répétitives en milieu industriel : ils explorent désormais des environnements non structurés, interagissent avec des objets et collaborent avec les humains. Parallèlement, l'émergence des grands modèles de langage (LLM) et des grands modèles vision-langage (VLM) a ouvert de nouvelles perspectives pour doter les robots de capacités de compréhension sémantique et de raisonnement, rendant possible la planification, la navigation et la manipulation d'objets à partir d'instructions en langage naturel.

1.1.1 Définitions clés

- LLM (Large Language Model) : modèle de deep learning entraîné sur de vastes corpus textuels, capable de générer et d'interpréter du langage humain avec un haut niveau de cohérence.
- VLM (Vision-Language Model) : extension des LLM, combinant traitement d'images et de texte pour associer des représentations visuelles à des concepts linguistiques, utile pour comprendre et décrire des scènes.
- Navigation autonome : capacité d'un robot à se déplacer dans un environnement inconnu ou partiellement connu, en construisant et exploitant des représentations spatiales pour atteindre des objectifs sans intervention humaine directe.

1.1.2 Problématique et objectifs du semestre

L'objectif de ce semestre était d'explorer l'exploitation d'un Large Vision Model pour la navigation autonome d'un robot. Après avoir étudié plusieurs solutions (MoMa-LLM, NaVid, VLMaps,), nous avons concentré nos efforts sur VLMaps, pour sa maturité et sa promesse de fusion vision-langage dans la création de cartes 3D exploitables. Toutefois, l'installation et la configuration du projet ont soulevé des défis techniques (compatibilité OS, dépendances GPU) que nous avons dû surmonter pour mettre en œuvre une boucle de navigation fonctionnelle.

1.1.3 Structure du rapport

Ce rapport s'articule en trois parties principales :

- État de l'art, présentant les grands projets LLM/VLM en robotique et la comparaison de leurs approches.
- Choix de la solution et obstacles rencontrés, décrivant notre sélection de VLMaps et les difficultés d'installation.
- Implémentation et résultats, détaillant la méthodologie, l'intégration de VLMaps et les expériences menées.

1.1.4 Démarche exploratoire et environnement technique

Le travail a été réalisé sur une machine personnelle fonctionnant sous Arch Linux, équipée d'un GPU NVIDIA 4070 Ti Super, ce qui a permis l'exécution efficace des différents modules et projets. Le développement s'est appuyé sur des bibliothèques essentielles telles que PyTorch, ROS, ainsi que l'ensemble des dépendances nécessaires au fonctionnement des projets explorés. Une attention particulière a été portée à la compatibilité entre les modules, à la gestion fine des environnements Python et aux contraintes spécifiques des pilotes GPU sous Linux.

Au cours du semestre, plusieurs pistes ont été explorées, notamment les projets MoMa-LLM, NaVid et d'autres frameworks de navigation sémantique. Chacun d'eux présentait un intérêt technique ou théorique particulier, mais des limitations importantes ont freiné leur mise en œuvre. Nous évoquerons les différentes voies explorées et les limitations techniques rencontrées.

Chapitre 2

État de l’art

2.1 Contexte

Cet état de l’art vise ainsi à mettre en lumière les progrès récents dans le domaine de la robotique, en se focalisant particulièrement sur l'application des grands modèles de langage (LLMs) et des grands modèles de vision-langage (VLMs). Ces technologies jouent désormais un rôle central dans divers aspects de la robotique, qu'il s'agisse de la prise de décision, de la planification, de la manipulation d’objets ou de la navigation intelligente.

2.2 Projets et publications

La littérature récente concernant l’application des LLMs et VLMs en robotique peut être structurée en quatre grands axes :

- Raisonnement : Applications des modèles pour l’analyse, l’explication et l’interprétation des processus décisionnels.
- Planification : Utilisation de l’IA pour la génération de stratégies et la conversion d’instructions en actions concrètes.
- Manipulation : Emploi de techniques avancées pour la manipulation d’objets et la coordination des actions robotiques.
- Instructions et Navigation : Intégration du langage naturel pour guider la navigation et la compréhension des environnements par les robots.

Une partie des publications peut être aussi liée au domaine de la robotique mais surtout de la simulation d'environnement et des benchmarks. Nous allons donc les traiter séparément.

2.3 Raisonnement

Projet	Titre	Code	Année
Gemini Robotics [1]	Bringing AI into the Physical World	–	2025
MoMa-LLM [2]	Language-Grounded Dynamic Scene Graphs for Interactive Object Search with Mobile Manipulation	GitHub	2024
ReKep [3]	Spatio-Temporal Reasoning of Relational Keypoint Constraints for Robotic Manipulation	GitHub	2024
AHA [4]	A Vision-Language-Model for Detecting and Reasoning over Failures in Robotic Manipulation	GitHub	2024

Table 2.1 – Projets de raisonnement et leurs caractéristiques.

Gemini Robotics

Présentation : Développé par Google DeepMind, Gemini Robotics repose sur le modèle Gemini 2.0 et intègre vision, langage et action pour contrôler directement des robots. Il exécute des mouvements fluides en s'adaptant à de nouveaux objets et environnements.

Atouts : intégration complète des modalités, réactivité, mouvements fluides. *Limites :* accès partiellement propriétaire, forte demande en ressources matérielles.

MoMa-LLM

Présentation : MoMa-LLM associe modèles de langage et graphes de scène dynamiques pour guider la recherche d'objets par des robots mobiles. L'approche sémantique permet une exploration intelligente d'environnements complexes.

Atouts : scènes dynamiques adaptatives, requêtes naturelles pour la localisation, utile en environnements vastes. *Limites :* dépendance à la qualité des données, implémentation complexe à maintenir.

ReKep

Présentation : ReKep génère des trajectoires de manipulation en boucle fermée grâce à une optimisation hiérarchique, en intégrant des contraintes spatio-temporelles via des modèles de vision et de langage.

Atouts : trajectoires précises, adaptation aux perturbations, robuste en environnement dynamique. *Limites :* complexité algorithmique, calcul coûteux en temps réel.

AHA

Présentation : AHA est un modèle open-source vision-langage dédié à la détection et à l'analyse des erreurs en manipulation robotique, avec retour en langage naturel.

Atouts : détection proactive d'échecs, feedback interprétable, amélioration continue. *Limites :* application très ciblée, performance liée à la qualité des descriptions en langage naturel.

Comparaison et domaines d'application

Gemini : contrôle direct et réactif pour des environnements variés.

MoMa-LLM : recherche d'objets sémantique en robotique mobile.

ReKep : précision de manipulation via modélisation spatio-temporelle.

AHA : retour d'expérience par détection d'erreurs.

Conclusion

Ces projets proposent des stratégies complémentaires : Gemini pour un comportement autonome intégré, MoMa-LLM pour la perception sémantique, ReKep pour des manipulations précises et AHA pour la détection d'erreurs. Le choix dépend des objectifs visés, des contraintes matérielles et logicielles (licences, compatibilité OS, puissance GPU), ainsi que de la complexité d'implémentation et de la disponibilité des ressources.

2.4 Planification

Projet	Titre	Code	Année
APD [5]	Interpretability in Parameter Space : Minimizing Mechanistic Description Length with Attribution-based Parameter Decomposition	–	2025
PSL [6]	Language Model Guided RL for Solving Long Horizon Robotics Tasks	GitHub	2024
Multi-Layer-LLMs [7]	Enhancing Robot Task Planning and Execution through Multi-Layer Large Language Models	–	2024

Table 2.2 – Vue d'ensemble des projets et de leur planification.

APD

Présentation : APD (Interpretability in Parameter Space) propose une décomposition des paramètres des réseaux neuronaux pour rendre leurs mécanismes internes plus transparents. Il s'appuie sur une attribution des composantes pour isoler des sous-ensembles interprétables et faciliter l'analyse des décisions.

Atouts : meilleure interprétabilité des modèles, identification des fonctions internes, utile pour audit et diagnostic. *Limites :* mise en œuvre complexe, difficile à étendre à grande échelle.

PSL

Présentation : PSL (Plan-Seq-Learn) associe langage naturel et apprentissage par renforcement pour exécuter des tâches robotiques longues. Il permet de traduire des instructions abstraites en commandes concrètes sans recourir à des bibliothèques de compétences prédéfinies.

Atouts : guidage par le langage, structure modulaire, pas de dépendance aux compétences préentraînées. *Limites :* dépendance à la qualité du modèle de langage, transfert simulation/réel difficile.

Multi-Layer LLMs

Présentation : Ce projet utilise des modèles de langage hiérarchiques pour planifier et exécuter des tâches complexes. La décomposition multi-niveaux des instructions permet une exécution adaptée à différents contextes robotiques.

Atouts : interprétation graduée des commandes, flexibilité face à la complexité des tâches, exécution plus efficace. *Limites :* coût computationnel élevé, difficulté d'intégration en temps réel dans des systèmes hybrides.

Comparaison et domaines d'application

APD : amélioration de l'interprétabilité des modèles, utile pour audit et analyse.

PSL : traduction du langage naturel en actions pour l'apprentissage par renforcement.

Multi-Layer LLMs : décomposition hiérarchique d'instructions pour planification/exécution fine.

Conclusion

Ces approches répondent à des enjeux complémentaires. APD améliore la lisibilité interne des modèles pour un meilleur contrôle. PSL connecte langage abstrait et actions concrètes via le RL pour des tâches longues. Multi-Layer LLMs structure les commandes complexes pour une exécution robuste et hiérarchisée.

2.5 Manipulation

Projet	Titre	Code	Année
LLM+MAP [8]	Bimanual Robot Task Planning using Large Language Models and Planning Domain Definition Language	GitHub	2025
Code-as-Monitor [9]	Constraint-aware Visual Programming for Reactive and Proactive Robotic Failure Detection	GitHub	2024
LABOR-Agent [10]	Large Language Models for Orchestrating Bimanual Robots	GitHub	2024

Table 2.3 – Projets de manipulation et leurs caractéristiques.

LLM+MAP

Présentation : LLM+MAP combine grands modèles de langage (LLM) et planification PDDL pour coordonner des tâches de manipulation bimanipulée. Il traduit des instructions en langage naturel en objectifs exploitables par des planificateurs robotiques, assurant une synchronisation fluide des deux bras.

Atouts : interprétation linguistique via LLM, planification rigoureuse avec PDDL, coordination efficace des bras. *Limites :* dépendance à la précision du langage, intégration technique complexe entre modules.

Code-as-Monitor

Présentation : Code-as-Monitor intègre des modèles vision-langage dans une programmation visuelle contrainte pour détecter et anticiper les défaillances robotiques. Le système surveille en temps réel les anomalies et guide la maintenance.

Atouts : surveillance proactive, programmation visuelle robuste, réactivité aux incidents. *Limites :* performance sensible à la qualité des contraintes définies, coût computationnel élevé.

LABOR-Agent

Présentation : LABOR-Agent utilise un LLM pour générer des politiques de contrôle bimanipulées dans des tâches longues. Il analyse les configurations et synchronise les actions des bras pour exécuter des missions complexes.

Atouts : génération de stratégies via LLM, coordination bras-bras, adaptation aux tâches évolutives. *Limites :* complexité des politiques, dépendance à la qualité du LLM.

Comparaison et domaines d'application

LLM+MAP : interprétation linguistique structurée via PDDL pour la manipulation coordonnée.

LABOR-Agent : orchestration complète de la manipulation via stratégie générée par LLM.

Code-as-Monitor : détection d'anomalies en robotique via vision, langage et programmation visuelle.

Conclusion

LLM+MAP et LABOR-Agent s'attaquent à la coordination bimanipulée via LLM, l'un par planification PDDL, l'autre par génération de stratégies. Code-as-Monitor se distingue en assurant la fiabilité des systèmes robotiques par une surveillance intelligente intégrant contraintes, vision et langage.

2.6 Instructions et Navigation

Projet	Titre	Code	Année
NaVid [11]	Video-based VLM Plans the Next Step for Vision-and-Language Navigation	GitHub	2024
OVSG [12]	Context-Aware Entity Grounding with Open-Vocabulary 3D Scene Graphs	GitHub	2023
VLMaps [13]	Vision-Language Maps for Robot Navigation	GitHub	2022

Table 2.4 – Projets d'instruction et de navigation et leurs caractéristiques.

NaVid

Présentation : NaVid est un modèle vision-langage basé sur la vidéo pour naviguer dans des environnements inconnus. Il utilise un flux vidéo monoculaire pour planifier les actions sans carte ni profondeur, avec un fort accent sur la généralisation à partir d'instructions linguistiques.

Atouts : approche sans carte, exploitation du langage naturel, bonne généralisation. *Limites :* traitement vidéo coûteux, sensibilité aux variations visuelles.

OVSG

Présentation : OVSG (Open-Vocabulary 3D Scene Graphs) permet l'ancrage d'entités dans des graphes de scène 3D à vocabulaire ouvert. Il relie objets et zones à des requêtes textuelles pour améliorer la compréhension spatiale dans des environnements complexes.

Atouts : requêtes textuelles flexibles, ancrage précis, aide à la navigation et manipulation. *Limites :* dépend de la qualité de la reconstruction 3D, difficile à adapter au temps réel.

VLMaps

Présentation : VLMaps fusionne modèles vision-langage et reconstruction 3D pour créer des cartes exploitables par des robots à partir d'instructions naturelles. Il génère une représentation spatiale riche pour guider la navigation.

Atouts : cartes 3D détaillées, compréhension des relations spatiales, navigation guidée par langage. *Limites :* dépendance aux modèles pré-entraînés, coût de la reconstruction 3D.

Comparaison et domaines d'application

NaVid : navigation sans carte à partir de flux vidéo, adaptée aux environnements inconnus.

OVSG : compréhension contextuelle via graphe 3D pour navigation ou manipulation guidée par texte.

VLMaps : fusion vision-langage et reconstruction 3D pour navigation sémantique précise.

Conclusion

NaVid améliore la navigation par apprentissage direct sur vidéo sans carte. OVSG structure l'environnement en graphe 3D pour répondre à des requêtes contextuelles. VLMaps génère des cartes spatiales sémantiques pour une navigation linguistique fine.

2.7 Simulation

Projet	Titre	Code	Année
OmniGibson [14]	A platform for accelerating Embodied AI research built upon NVIDIA's Omniverse engine	GitHub	2024
iGibson [15]	A Simulation Environment to train Robots in Large Realistic Interactive Scenes	GitHub	2024

Table 2.5 – Projets de simulation et leurs caractéristiques.

OmniGibson est une plateforme modulable construite sur NVIDIA Omniverse, conçue pour accélérer la recherche en intelligence artificielle incarnée grâce à des environnements virtuels photoréalistes, dynamiques et entièrement personnalisables.

iGibson fournit un cadre de simulation interactif à grande échelle, permettant d'entraîner et de valider des robots sur des tâches complexes de perception et de manipulation dans des scènes réalistes du quotidien.

Les deux projets intègrent une modélisation physique avancée, une détection de collisions précise et un rendu haute fidélité, essentiels pour transférer efficacement les politiques acquises en simulation vers des robots réels.

Ils offrent des outils de benchmarking standardisés, favorisant la comparaison et l'optimisation d'algorithmes de navigation autonome, de préhension d'objets et de planification de trajectoire.

Grâce à leur architecture open source et à une communauté active, OmniGibson et iGibson évoluent en continu, intégrant de nouveaux capteurs et extensions robotiques.

Leur adoption massive réduit significativement les coûts, les délais et les risques associés aux expérimentations physiques, tout en améliorant la robustesse et la sécurité des systèmes robotiques avant leur déploiement réel.

Chapitre 3

Choix de la solution et obstacles rencontrés

Dans le cadre de ce travail, nous avons essayé plusieurs projets pour trouver des cas d'applications réelles à ces projets. Dans cette section, nous allons expliquer quel est le projet que nous avons utilisé et pourquoi nous avons fait ce choix. Nous aborderons également les différents obstacles rencontrés lors de la mise en place des différents projets et les raisons de ces difficultés. Le choix des projets s'est fait de manière assez simple : il s'agissait de sélectionner un projet à faire fonctionner, puis de l'utiliser sur un robot en simulation ou sur un robot réel. Nos premiers choix se sont portés sur les projets de raisonnement, car ils me paraissaient les plus intéressants à mettre en place et à comprendre. Ces projets présentaient un fort potentiel en termes de raisonnement sémantique, d'interprétation des instructions en langage naturel et de prise de décision autonome.

3.1 Raisonnement

Pour les projets testés, nous avons choisi des projets open source avec un code accessible au public et avons décidé de tester les projets ayant le plus d'étoiles sur GitHub. Cette approche nous a permis de nous concentrer sur des solutions déjà éprouvées et appréciées par la communauté, augmentant ainsi nos chances de succès et de compatibilité avec nos besoins.

3.1.1 MoMa-LLM

Après quelques ratés dus à des problèmes d'installation du projet, nous avons finalement réussi à le faire fonctionner. Le principal problème était lié aux différentes dépendances du projet. Après de nombreux tests, nous avons réussi à le faire fonctionner dans un conteneur Docker créé par l'équipe du projet. Après avoir testé différentes fonctions du projet, nous nous sommes rendu compte que le projet était sous licence et qu'il nous était impossible de l'utiliser en l'état. Cela nous a conduits à explorer d'autres solutions ou à envisager de contacter les propriétaires du projet pour discuter des possibilités de collaboration ou d'obtention d'une licence appropriée. Cette expérience nous a permis de mieux comprendre l'importance de vérifier les licences dès le début d'un projet pour éviter des complications futures.

Le projet initial a été développé il y a plusieurs années, ce qui s'est traduit par l'utilisation de versions désormais obsolètes de plusieurs bibliothèques et outils clés. Par exemple, le code cible Python 3.6 tandis que nos environnements modernes tournent sous Python 3.8 ou 3.9, entraînant des incompatibilités syntaxiques (f-strings différentes, gestion des annotations de type, API de asyncio évoluée, etc.). De nombreuses dépendances citées dans le requirements.txt pointent vers des versions intermédiaires de NumPy, SciPy ou PyTorch qui ne sont plus maintenues sur PyPI, provoquant des erreurs du type « no matching distribution ». En dehors de Python, plusieurs modules C/C++ font appel à des symboles qui ont changé dans glibc ou dans le compilateur GCC — sans parler des appels directs à des fonctions internes de CUDA 10.2, désormais retirées ou déplacées dans CUDA 11.x.

Le conteneur Docker utilisé était celui fourni dans le dépôt officiel, basé sur Ubuntu 20.04 avec CUDA 11.7. Nous avons pu tester les capacités de navigation sémantique et de recherche d'objet par requêtes textuelles. Toutefois, certaines parties critiques du code ou du modèle étaient restreintes par une licence non open-source (par exemple : modèle MoMa inaccessible sans demande explicite), ce qui bloquait son exploitation complète.

3.1.2 ReKep

Les problèmes de fonctionnement de ReKep ne sont pas venus directement du projet lui-même. Le projet a été conçu pour fonctionner sur OmniGibson. La principale problématique résidait dans l'installation d'OmniGibson sur Arch Linux. Comme le projet n'est pas officiellement supporté sur cette distribution, nous avons rencontré quelques difficultés à faire fonctionner le simulateur sur notre machine.

Après avoir rencontré l'impossibilité d'exécuter OmniGibson sur Arch Linux — la distribution n'étant pas officiellement prise en charge — nous avons tenté une migration vers Windows. Or, cet environnement ne disposait d'aucun prérequis pour le développement, nous obligeant à installer manuellement chaque composant de base (CMake, Visual Studio Build Tools, pilotes NVIDIA, etc.), sans pour autant parvenir à faire fonctionner ReKep. Après de nombreux échecs et allers-retours entre Windows et Arch Linux, nous avons finalement décidé d'abandonner l'installation de ReKep sur ces systèmes. Le blocage est clairement dû au manque de support d'OmniGibson pour la distribution utilisée.

Après ces différents problèmes, nous nous sommes tournés vers des projets de navigation. Étant plus simples à mettre en place dans une simulation ou dans un environnement réel, et ayant un lien direct avec des problématiques comme la voiture autonome, nous avons jugé utile d'essayer un projet concentré sur la navigation par IA générative. De plus, les projets de navigation bénéficient souvent d'une communauté plus large, d'une documentation plus fournie et de benchmarks standardisés, ce qui facilite leur expérimentation en simulation.

3.2 Navigation

3.2.1 Navid

Pour Navid, le problème était similaire à celui rencontré avec ReKep. Le projet a été développé sur une version très ancienne de Habitat-sim, ce qui nous a rendu impossible l'exécution de Habitat-sim, et par extension de Navid, sur notre machine. Malgré plusieurs tentatives de rétrocompatibilité et d'adaptation du code, les incompatibilités entre les versions et les dépendances ont persisté. Cela nous a conduits à envisager d'autres solutions, comme la recherche de versions alternatives du projet ou l'exploration de simulateurs plus modernes et mieux maintenus pour nos besoins en matière de navigation et de simulation.

NaVid repose sur Habitat-sim v0.1.7, une version obsolète et incompatible avec les systèmes récents (comme Python ≥3.10, ou CUDA >11.3). Navid est un projet récent avec beaucoup de possibilités futur. Il est sûr qu'avec un peu plus de temps il aurait été possible de finalement faire fonctionner Habitat-sim et de faire NaVid tourner sur notre machine.

Une mise à jour de Habitat-sim vers une version plus récente serait possible pour garantir la compatibilité et exploiter pleinement les optimisations offertes par les systèmes actuels. Cette évolution technique permettrait d'améliorer la stabilité du projet et d'ouvrir de nouvelles perspectives pour le développement de fonctionnalités avancées.

3.2.2 VLMaps

Après de nombreux essais, nous avons réussi à faire fonctionner toutes les fonctionnalités de VLMaps. Ces fonctionnalités ont été émulées directement depuis le projet, en utilisant la partie démonstration. La prochaine étape consiste à intégrer VLMaps dans un système réaliste. L'idée est de suivre le schéma présenté dans la figure 3.1 et d'utiliser VLMaps dans une simulation afin de créer

une preuve de concept. Cela nous permettra de démontrer la faisabilité de l'installation et de l'utilisation du projet dans un environnement réel. Cette étape est cruciale pour valider l'efficacité et la robustesse de VLMaps dans des conditions opérationnelles, ouvrant ainsi la voie à des applications pratiques et innovantes dans le domaine de la cartographie et de la navigation autonome.

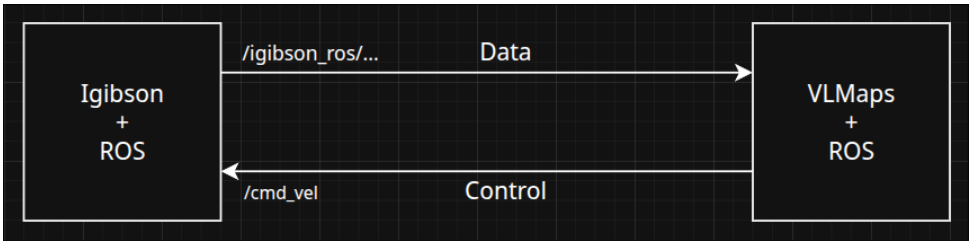

Figure 3.1 – Objectif d’application de VLMaps avec iGibson

Chapitre 4

Méthodologie et installation

4.1 Configuration de l'environnement

Afin de garantir une isolation stricte et une portabilité maximale de nos outils de cartographie et de simulation, nous avons adopté une approche fondée sur la conteneurisation. Cette démarche nous permet non seulement d'uniformiser les environnements de développement et de production, mais aussi de simplifier le déploiement sur différentes machines hôtes (ici, Arch Linux) sans compromettre la compatibilité des pilotes GPU, des serveurs d'affichage ou des dépendances ROS. En orchestrant deux conteneurs distincts (l'un dédié à VLMaps et l'autre à iGibson), nous obtenons un découplage net entre la génération et le rendu des cartes 3D et les simulations robotisées, tout en facilitant la montée en charge, la mise à jour indépendante de chaque brique et la reproduction fidèle des expérimentations.

Après avoir testé VLMaps sur Arch Linux pour évaluer ses fonctionnalités, nous avons décidé de l'isoler dans un conteneur Docker afin de garantir un environnement reproductible et de simplifier le déploiement. Nous avons construit une image Docker basée sur Ubuntu 20.04, dans laquelle nous avons installé Miniconda. Dans cette image, nous avons créé un environnement VLMaps sous Python 3.8 et y avons installé toutes les dépendances nécessaires à VLMaps.

Pour permettre à VLMaps d'afficher ses interfaces graphiques, nous avons configuré le conteneur pour qu'il accède au serveur X11 et à Wayland de l'hôte, ainsi qu'au GPU via les pilotes NVIDIA et le nvidia-container-runtime. Bien que la machine hôte tourne sous Arch Linux, ce choix d'un conteneur Ubuntu 20.04 nous offre une compatibilité maximale et une isolation complète entre l'environnement de développement et le système de production.

Une fois VLMaps conteneurisé, nous y avons également installé toutes les dépendances nécessaires à ROS. Nous avons ensuite créé un second conteneur, basé sur iGibson_ros, pour notre déploiement. Après quelques ajustements pour que ce conteneur fonctionne correctement sur Arch Linux et pour garantir l'affichage de l'environnement ROS dans RViz, nous avons pu commencer à préparer les différents modules permettant la communication entre le conteneur VLMaps et le conteneur iGibson.

4.2 Installation de ROS et des packages nécessaires

Afin de piloter VLMaps et iGibson via ROS Noetic sur Ubuntu 20.04, nous avons suivi deux guides : l'installation officielle de ROS Noetic et l'intégration ROS d'iGibson.

Étape 1 : Installation de ROS Noetic (ros-base)

1. `sudo apt update && sudo apt install -y curl`
2. `sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu focal main" > /etc/apt/sources.list.d/ros-latest.list'`
3. `curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc`
4. `sudo apt-key add -`
5. `sudo apt update`
6. `sudo apt install -y ros-noetic-ros-base`
7. `echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc`
8. `source ~/.bashrc`

Étape 2 : Workspace ROS pour VLMaps

1. `mkdir -p ~/catkin\_ws/src && cd ~/catkin\_ws`
2. `catkin\_make`
3. `echo "source ~/catkin\_ws/devel/setup.bash" >> ~/.bashrc`
4. `echo "export ROS\_MASTER\_URI=http://iGibson\_main\_test:11311" >> ~/.bashrc`
5. `source ~/.bashrc`

Étape 3 : Workspace ROS pour iGibson

1. `mkdir -p ~/catkin\_ws/src`
2. `ln -s <iGibson\_root>/examples/ros/iGibson-ros~/catkin\_ws/src/iGibson\_ros`
3. `cd ~/catkin\_ws`
4. `rosdep update`
5. `rosdep install --from-paths src --ignore-src -r -y`
6. `catkin\_make`
7. `echo "source ~/catkin\_ws/devel/setup.bash" >> ~/.bashrc`
8. `echo "export ROS\_MASTER\_URI=http://iGibson_main_test:11311" >> ~/.bashrc`
9. `source ~/.bashrc`

Étape 4 : Vérification

1. `roscore &`
2. `roslaunch iGibson\_ros turtlebot\_rgbd.launch`

Le lancement de `turtlebot_rgbd.launch` valide le bon fonctionnement de ROS, du bridge VLMaps et de l'intégration iGibson.

4.2.1 Mise en place du réseau inter-conteneurs

Pour connecter deux conteneurs Docker, on commence par leur fournir un réseau virtuel commun sur lequel ils pourront s'adresser directement l'un à l'autre. Concrètement, on crée un réseau Docker dédié, qui agit comme un commutateur virtuel isolé, on attache chacun des conteneurs à ce même réseau. Une fois sur ce réseau partagé, chaque conteneur se voit attribuer un nom d'hôte interne correspondant à son nom de service ou de conteneur, et peut donc communiquer avec ses pairs en utilisant simplement ce nom plutôt que des adresses IP. Cette approche garantit une isolation réseau (les conteneurs ne sont pas exposés à l'extérieur) tout en assurant une communication bidirectionnelle fluide et dynamique entre les services.

4.3 Lancement des environnements

Pour tester les environnements et s'assurer qu'ils communiquent correctement, nous lançons simultanément les deux masters ROS, l'un dans le conteneur iGibson et l'autre dans le conteneur

VLMaps. Il suffit ensuite de configurer les variables d'environnement `ROS_MASTER_URI` et `ROS_IP` de chaque conteneur pour qu'ils se pointent mutuellement.

Une fois ces réglages en place, nous vérifions la liaison en publiant et en souscrivant à quelques topics partagés, par exemple `/gibson_ros/camera/rgb/image` pour l'image RGB et `/cmd_vel` pour les commandes de déplacement. Si les messages transitent sans perte et que les deux conteneurs réagissent aux commandes l'un de l'autre, cela confirme que le réseau inter-conteneurs et la configuration ROS fonctionnent comme prévu.

4.3.1 Lancement d'iGibson

Après avoir configuré et compilé le workspace ROS dans le conteneur iGibson, nous avons démarré le master ROS (« roscore ») puis lancé la simulation de démo fournie dans le dépôt GitHub d'iGibson (`roslaunch igibson_ros turtlebot_rgbd.launch`). Cette étape nous permet de vérifier que le simulateur fonctionne correctement et publie bien les topics essentiels (capteurs, état du robot, etc.).

4.3.2 Lancement de VLMaps

Dans le conteneur VLMaps, nous avons simplement utilisé `rostopic list` pour nous assurer que les différents topics de iGibson était visible depuis le conteneur VLMaps.

Ensuite, une fois les deux environnements ROS en route, nous pouvons procéder aux premiers tests d'intégration en publiant des commandes de navigation depuis VLMaps et en observant leur effet dans iGibson. Cela nous permet de valider la communication inter-conteneurs et la cohérence des topics partagés avant d'attaquer des scénarios de navigation plus complexes.

4.4 Tests et validation

Pour tester la bonne communication entre les deux conteneurs, nous avons utilisé un simple script ROS qui récupère les images publiées sur le topic `/gibson_ros/camera/rgb/image` et publie simultanément des commandes sur le topic `/reset_pose`. En visualisant le flux d'images et en observant le repositionnement du robot dans iGibson après chaque publication de `/reset_pose`, nous avons pu confirmer que :

- VLMaps reçoit bien les données visuelles provenant de la simulation iGibson.
- Les commandes émises par VLMaps sur `/reset_pose` sont correctement interprétées et appliquées par le conteneur iGibson.

Cette validation simple mais efficace garantit que le pont ROS entre les deux conteneurs fonctionne correctement avant de passer à des scénarios de navigation plus complexes.

4.5 Résumé de la méthodologie

Dans cette partie, nous avons présenté une chaîne de traitement complète, de l'environnement de développement jusqu'à la validation de la communication inter-conteneurs. Nous avons d'abord mis en place deux conteneurs Docker – l'un dédié à VLMaps, l'autre à iGibson – pour garantir une isolation forte et une portabilité maximale. Chaque conteneur repose sur Ubuntu 20.04 et intègre les dépendances spécifiques (Miniconda pour VLMaps, build ROS Noetic et packages iGibson pour le second). Nous avons ensuite configuré un réseau Docker privé, permettant aux deux instances ROS de se découvrir et d'échanger via des variables d'environnement partagées. Le lancement coordonné des simulateurs (iGibson) et des services de cartographie (VLMaps) a été validé grâce à des commandes de test ROS simples (`rostopic list`, `rostopic echo`, `roslaunch`), qui ont confirmé la réception des images, la publication des cartes et l'exécution de commandes de déplacement. Cette méthodologie garantit une réplication fiable de l'expérimentation et prépare le terrain pour les scénarios de navigation autonome à venir.

Chapitre 5

Expériences et résultats

L'objectif principal était de vérifier si VLMaps pouvait être intégré dans un environnement réel pour générer une carte sémantique. Pour cela, nous avons créé une scène de simulation sous iGibson afin d'explorer l'environnement virtuel, d'y collecter les données nécessaires à VLMaps pour construire la carte, puis de déplacer le robot au sein de cette scène en utilisant VLMaps.

5.1 Protocole expérimental

Notre objectif était de tester l'intégration de VLMaps dans un environnement « réel » simulé. Pour cela, nous avons d'abord créé une scène de simulation simple sous iGibson, puis fait explorer un robot TurtleBot à l'aide de VLMaps pour générer une carte sémantique et piloter sa navigation.

Description de la scène

La scène se compose de quatre pièces aménagées :

- Un salon, équipé d'un téléviseur et d'un tableau mural.
- Une salle d'eau, avec toilettes et douche.
- Une cuisine, comprenant un plan de travail et des meubles de rangement.
- Une chambre, avec lit et mobilier.

Étapes de l'expérience

1. **Déploiement du robot** Nous plaçons le TurtleBot au point de départ de la scène simulée.
2. **Collecte de données** Pendant l'exploration, nous enregistrons à chaque image :
    - La pose du robot (position et orientation).
    - L'image RGB capturée par la caméra frontale.
    - La carte de profondeur associée.
3. **Génération de la carte sémantique** À partir de ces données, VLMaps construit une carte sémantique 3D de l'environnement.
4. **Navigation automatisée** Nous exécutons un script ROS qui consomme la carte générée pour envoyer des commandes de déplacement au robot, validant ainsi la boucle complète exploration – cartographie – navigation.

5.2 Collecte et traitement des données

Pour explorer notre scène, nous avons procédé manuellement, en parcourant pièce par pièce chaque zone afin de collecter les données nécessaires à la carte. Nous avions défini un trajet d'exploration couvrant systématiquement le salon, la salle d'eau, la cuisine et la chambre, de sorte à garantir une couverture maximale.

La collecte de données dans la scène fut très simple grâce au script présenté en annexe (A), qui se connecte au topic ROS d'iGibson et sauvegarde automatiquement chaque image et chaque profondeur dans la structure de dossiers attendue par VLMaps.

Concrètement, pour chaque instant t :

- L'image RGB est capturée puis enregistrée sous forme de fichier PNG numéroté (`000000.png`, `000001.png`, …) dans le dossier `rgb`.
- La carte de profondeur correspondante est extraite du flux et sauvegardée au format NumPy (`.npy`) dans le dossier `depth`.
- La pose du robot (position et orientation dans le repère de la scène) est appendue ligne par ligne dans `poses.txt`, au format `x y z qx qy qz qw`.

Cette organisation reproduit exactement la structure imposée par VLMaps pour les données d'entrée :

```
vlmaps_data_dir/
 customized_scene_1/
      rgb/
          000000.png
          000001.png
          …
      depth/
          000000.npy
          000001.npy
          …
      poses.txt
```

Figure 5.1 – Exemple d'image RGB de la scène simulée

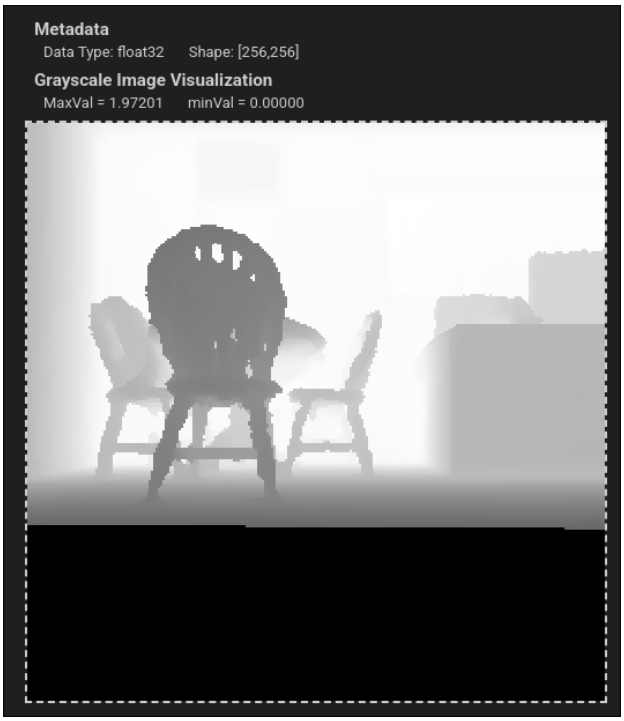

Figure 5.2 – Exemple d'image de profondeur de la scène simulée

Grâce à ce format standardisé, VLMaps peut directement charger les séquences d'images et de profondeurs, ainsi que les poses associées, pour :

1. reconstruire une carte 3D probabiliste de l'environnement,
2. y superposer les couches sémantiques (via segmentation d'images ou annotations),

3. et enfin générer une map sémantique exploitable pour la navigation autonome.

Toutes ces étapes s'opèrent en batch, sans intervention manuelle, ce qui garantit la reproductibilité de la collecte et simplifie le pipeline d'entraînement ou d'évaluation dans un contexte réel ou simulé.

5.3 Scénarios de navigation

En raison de contraintes de temps et de la nécessité de finaliser le projet rapidement, la partie relative aux scénarios de navigation n'a pas encore été entièrement développée. Les tests d'exploration libre, de suivi de trajectoire et d'évitement d'obstacles restent à tester pour VLMaps en situation simulé ou réelle. Cette étape sera menée dans les semaines à venir afin de compléter l'évaluation de VLMaps en conditions réalistes et de valider pleinement son intégration pour la navigation autonome.

5.4 Résultats et conclusion

À ce stade, les expériences de génération de carte sémantique et de navigation automatisée sont en cours. Voici une conclusion préliminaire au résultat que nous avons déjà et ce que nous avons réussi à acomplir.

5.4.1 Limitations et échecs

Plusieurs facteurs ont retardé l'aboutissement de ce projet. Tout d'abord, des expérimentations préliminaires (NavID, MoMA-LLM, etc.) n'ont pas fonctionné comme prévu, ce qui a mobilisé un temps considérable sans produire de résultats exploitables. Ensuite, la configuration des conteneurs a pris plus de temps que prévu, en raison notamment de problèmes liés à ma distribution (Arch Linux) et à mon gestionnaire de fenêtres (Hyprland). Le fonctionnement de Hyprland est basé sur Wayland. Wayland est un petit peu différent de X11, le temps de comprendre ces différences et les besoins différents demandés par Docker pour faire fonctionner des applications graphiques sur notre machine nous a retardé dans le déploiement des contenaires.

Sur le plan de la recherche, il est encore prématuré de tirer des conclusions définitives. Cependant, VLMaps s'est révélé être un projet intéressant et prometteur, facile à mettre en œuvre une fois les dépendances installées. La démonstration fournie dans le dépôt explique clairement les différentes étapes de traitement et met en évidence l'articulation de l'outil autour de l'IA générative. VLMaps permet de générer des cartes sémantiques personnalisées à partir d'environnements déjà scannés, ce qui laisse entrevoir des applications concrètes, par exemple comme moteur de planification pour un robot aspirateur capable de prioriser certaines zones d'une habitation.

5.4.2 Prochaines étapes

Les prochaines phases du projet consistent à :

- Intégrer la carte générée dans VLMaps ;
- Expérimenter la génération sémantique et les algorithmes de navigation du robot au sein de cette carte ;
- Valider ces étapes pour pouvoir formuler des conclusions fiables sur l'efficacité et l'intégration de VLMaps dans un environnement réel.

Ces travaux sont essentiels pour évaluer pleinement les capacités de VLMaps et envisager ses applications pratiques.

Chapitre 6

Conclusion et Bilan personnel

6.1 Conclusion

Comme évoqué précédemment, de nombreux projets ont été étudiés dans le cadre de TY22. Par manque de temps, nous n'avons malheureusement pas pu mener à bien l'intégration d'un projet dans un environnement réel. L'idée initiale était de sélectionner un projet et de le déployer directement sur un robot dans une scène réelle, mais pour des raisons techniques nous avons reporté cette phase sur la simulation. Celle-ci s'est avérée plus simple à mettre en place pour effectuer les tests nécessaires au bon fonctionnement des différents modules explorés. Après avoir choisi VLMaps, nous avons donc concentré nos efforts sur son déploiement dans iGibson, mais cette partie reste encore à finaliser.

6.2 Bilan personnel

Même si le projet n'a pas été menée a son terme en temps et en heure. Ce projet à été très interessant et m'a permis d'en apprendre beaucoup plus sur le fonctionnement de ROS et son implémentation en robotique. Au dela de ROS, j'ai pu appercevoir les travaux menée autour de la robotique, particulièrement sur l'utilisation de VLM et LLM dans ce domaine de recherche. J'ai pu comprendre dans quelle cadre l'intelligence artificielle était employer et quelles problématiques la recherche actuelle essaie de résoudre. Je considère un petit peu ce travail comme un échec car il n'y a pas beaucoup de résultat exploitable. Mais c'est grâce a tout ces ratés, impossibilité a faire fonctionné certain projet, d'autre limiation lié a des liscences ou encore des problématique lié a ma distribution, que j'ai pu en apprendre autant. Je ne savais pas comemnt X11 fonctionnais, maintenant je sais. Je ne savais pas comment fonctionnais ROS et pourquoi il était utiliser, maintenant j'en ai une idée plus claire. Alors même si d'un point de vu critique, le travail n'a pas été menée a bien en temps et en heure, il a été très instructif et j'en tirerais des leçons pour mes prochains travaux.

Bien que le projet n'ait pas été mené à son terme dans les délais impartis, il a été particulièrement enrichissant. J'ai considérablement approfondi ma compréhension de ROS et de son rôle essentiel en robotique. Au-delà de ROS, j'ai découvert les recherches actuelles portant sur l'application des VLM et des LLM en robotique, ce qui m'a permis de mieux saisir les problématiques que la communauté scientifique cherche à résoudre.

Je reconnais qu'il n'existe pas encore de résultats pleinement exploitables, et je pourrais considérer ce travail comme un échec au sens où l'objectif initial n'a pas été atteint. Cependant, toutes les difficultés rencontrées — impossibilité de faire fonctionner certains projets, contraintes de licences, problèmes liés à ma distribution Linux — ont été autant d'occasions d'apprentissage. Je suis désormais capable de comprendre le fonctionnement de X11 pour l'affichage graphique dans un conteneur, de comprendre l'utilisation de ROS et comment l'implémenter dans un outil de simulation ;

Alors même si d'un point de vu critique, le travail n'a pas été menée à bien en temps et en heure, il a été très instructif et j'en tirerais des leçons pour mes prochains travaux.

Bibliographie

[1] Google DeepMind Team. *Gemini Robotics : Bringing AI into the Physical World*, arXiv preprint arXiv :2503.20020. Vision–Language–Action generalist model controlling real robots ; publié sur arXiv le 12 mars 2025. 2025.

[2] Daniel Honerkamp, Martin Büchner, Fabien Despinoy, Tim Welschehold, et Abhinav Valada. « Language□Grounded Dynamic Scene Graphs for Interactive Object Search with Mobile Manipulation ». In : *IEEE Robotics and Automation Letters* 2024.

[3] Wenlong Huang, et al.. *ReKep : Spatio□Temporal Reasoning of Relational Keypoint Constraints for Robotic Manipulation*, GitHub repository. Implémentation officielle intégrée dans OmniGibson. 2024.

[4] Jiafei Duan, Wilbert Pumacay, Nishanth Kumar, Yi Ru Wang, Shulin Tian, Wentao Yuan, Ranjay Krishna, Dieter Fox, Ajay Mandlekar, et Yijie Guo. *AHA : A Vision□Language□Model for Detecting and Reasoning over Failures in Robotic Manipulation*, GitHub repository. Modèle open□source NVLabs pour détection/analyse d'échecs. 2024.

[5] Dan Braun, Lucius Bushnaq, Stefan Heimersheim, Jake Mendel, et Lee Sharkey. « Interpretability in Parameter Space : Minimizing Mechanistic Description Length with Attribution□based Parameter Decomposition ». In : *arXiv preprint arXiv :2501.14926* 2025.

[6] Murtaza Dalal, Tarun Chiruvolu, Devendra Chaplot, et Ruslan Salakhutdinov. « Plan□Seq□Learn : Language Model Guided RL for Solving Long Horizon Robotics Tasks », In : *Proceedings of ICLR 2024*. Approche modulaire LLM + motion planning + RL ; code disponible. 2024.

[7] Zhirong Luan, Yujun Lai, Rundong Huang, Shuanghao Bai, Yuedi Zhang, Haoran Zhang, et Qian Wang. « Enhancing Robot Task Planning and Execution through Multi□Layer Large Language Models ». In : *Sensors* 2024. doi : 10.3390/s24051687.

[8] Kun Chu, Xufeng Zhao, Cornelius Weber, et Stefan Wermter. *LLM+MAP : Bimanual Robot Task Planning using Large Language Models and Planning Domain Definition Language*, arXiv preprint arXiv :2503.17309. Code disponible sur GitHub. 2025.

[9] Enshen Zhou, Qi Su, Cheng Chi, Zhizheng Zhang, Zhongyuan Wang, Tiejun Huang, Lu Sheng, et He Wang. « Code□as□Monitor : Constraint□aware Visual Programming for Reactive and Proactive Robotic Failure Detection ». In : *arXiv preprint arXiv :2412.04455* 2024.

[10] Kun Chu, Xufeng Zhao, Cornelius Weber, Mengdi Li, Wenhao Lu, et Stefan Wermter. *Large Language Models for Orchestrating Bimanual Robots*, arXiv preprint arXiv :2404.02018. Code disponible sur GitHub. 2024.

[11] Jiazhao Zhang, Kunyu Wang, Rongtao Xu, Gengze Zhou, Yicong Hong, Xiaomeng Fang, Qi Wu, Zhizheng Zhang, et He Wang. « NaVid : Video□based VLM Plans the Next Step for Vision□and□Language Navigation ». In : *Proceedings of Robotics : Science and Systems* 2024.

[12] Haonan Chang, Kowndinya Boyalakuntla, Shiyang Lu, Siwei Cai, Eric Pu Jing, Shreesh Keskar, Shijie Geng, Adeeb Abbas, Lifeng Zhou, Kostas E. Bekris, et Abdeslam Boularias. « Context□Aware

Entity Grounding with Open□Vocabulary 3D Scene Graphs », In : *7th Annual Conference on Robot Learning (CoRL)*. Acceptée à CoRL 2023, code disponible sur GitHub. 2023.

[13] Chenguang Huang, Oier Mees, Andy Zeng, et Wolfram Burgard. « Visual Language Maps for Robot Navigation », In : *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. arXiv preprint arXiv :2210.05714 ; code disponible sur GitHub. London, United Kingdom, 2023. doi : 10.1109/ICRA48891.2023.10160969.

[14] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, et et al.. « BEHAVIOR-1K : A Human-Centered, Embodied AI Benchmark with 1,000 Everyday Activities and Realistic Simulation », In : *CoRL*. Introduit OmniGibson, construit sur NVIDIA Omniverse et PhysX 5 ; supporte rigides, fluides, déformables et états physiques avancés :contentReference[oaicite :1]index=1. 2022. Disponible sur : https://github.com/StanfordVL/BEHAVIOR-1K.

[15] Bokui Shen, Fei Xia, Chengshu Li, Roberto Martín-Martín, Linxi Fan, Guanzhi Wang, Claudia Pérez-D'Arpino, Shyamal Buch, Sanjana Srivastava, Lyne Tchapmi, Micael Tchapmi, Kent Vainio, Josiah Wong, Li Fei-Fei, et Silvio Savarese. « iGibson 1.0 : A Simulation Environment for Interactive Tasks in Large Realistic Scenes ». In : *arXiv preprint arXiv :2012.02924* 2020. Disponible sur : https://github.com/StanfordVL/iGibson.

Annexe A

Listing A.1 – Recorder ROS – script complet

```
#!/usr/bin/env python3
import sys
import os
import rospy
import cv2
import numpy as np
from sensor_msgs.msg import Image
from nav_msgs.msg import Odometry
from cv_bridge import CvBridge

class Recorder:
    def __init__(self, out_dir):
        self.bridge = CvBridge()
        self.out_dir = out_dir
        os.makedirs(os.path.join(out_dir, 'rgb'), exist_ok=True)
        os.makedirs(os.path.join(out_dir, 'depth'), exist_ok=True)
        self.pose_file = open(os.path.join(out_dir, 'poses.txt'), 'w')
        self.count = 0

        rospy.loginfo("[Recorder] Subscribing to RGB …topic")
        rospy.Subscriber('/gibson_ros/camera/rgb/image', Image, self.
   rgb_callback)

    def rgb_callback(self, msg_rgb):
        idx = f"{self.count:06d}"

        cv_img = self.bridge.imgmsg_to_cv2(msg_rgb, 'bgr8')
        cv2.imwrite(os.path.join(self.out_dir, 'rgb', f"{idx}.png"), cv_img)

        try:
            msg_depth = rospy.wait_for_message(
                '/gibson_ros/camera/depth/image', Image, timeout=2.0)
            msg_odom  = rospy.wait_for_message(
                '/odom', Odometry, timeout=2.0)
        except rospy.ROSException as e:
            rospy.logwarn(f"[Recorder] Timeout waiting for depth or odom: {e}")
            return

        dp = self.bridge.imgmsg_to_cv2(msg_depth, desired_encoding='passthrough
   ')
        np.save(os.path.join(self.out_dir, 'depth', f"{idx}.npy"), dp)

        p = msg_odom.pose.pose.position
        q = msg_odom.pose.pose.orientation
        line = (
            f"{idx} "
            f"{p.x:.6f} {p.y:.6f} {p.z:.6f} "
            f"{q.x:.6f} {q.y:.6f} {q.z:.6f} {q.w:.6f}\n"
        )
        self.pose_file.write(line)

        self.count += 1
```

```
        rospy.loginfo(f"[Recorder] Saved frame {idx}")

    def spin(self):
        rospy.spin()
        self.pose_file.close()

if __name__ == '__main__':
    rospy.init_node('scene_recorder', anonymous=True, log_level=rospy.INFO)

    if len(sys.argv) < 2:
        rospy.logerr("Usage: extract_igibson.py <output_dir>")
        sys.exit(1)
    out_dir = sys.argv[1]

    Recorder(out_dir).spin()
```

Exploitation d'un Large Vision Model (LVM) pour la navigation autonome d'un robot.

Ce rapport explore l'utilisation d'un Large Vision Model (LVM) pour la navigation autonome d'un robot, en se concentrant sur l'intégration de VLMaps dans un environnement simulé. L'objectif principal était de valider la faisabilité de l'utilisation de VLMaps pour générer des cartes sémantiques exploitables par un robot dans un environnement inconnu. Le document commence par une introduction au contexte général de l'utilisation des modèles de langage et de vision dans la robotique, soulignant les avancées récentes et les défis associés. Il présente ensuite un état de l'art des projets et publications récents dans le domaine, structuré autour de quatre axes principaux : raisonnement, planification, manipulation et navigation.
Le rapport détaille les différentes solutions explorées, notamment MoMa-LLM, ReKep, Navid et VLMaps, ainsi que les défis techniques rencontrés lors de leur installation et configuration. La méthodologie et l'installation sont décrites en détail, incluant la configuration de l'environnement, l'installation de ROS et des packages nécessaires, ainsi que le lancement des environnements de simulation. Bien que les scénarios de navigation n'aient pas été entièrement développés en raison de contraintes de temps, le rapport conclut avec un bilan personnel, soulignant les apprentissages tirés des défis rencontrés et les perspectives futures pour le projet.

Université de Technologie de Belfort-Montbéliard
4 Rue Edouard Branly, 90000 Belfort
www.utbm.fr