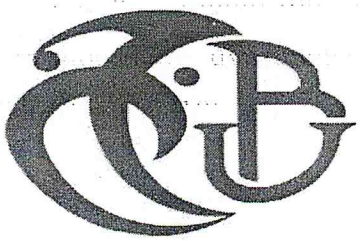
République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Saad Dahlab Blida

N° D'ordre :.....



Faculté des sciences

Département d'informatique

Mémoire Présenté par :

Ammari Naziha Madoui Roumaissaa

En vue d'obtenir le diplôme de master

Domaine : Mathématique et informatique

Filière : Informatique
Spécialité : Informatique
Option : Ingénierie de logiciel

Sujet :

Développement d'un Simulateur 3D pour système Multi-robots hétérogènes.

Soutenu le :

M.	Président
M.	Examinateur
M.	Examinateur
Mme. Oukid Saliha	Promotrice
Mme. Akli Isma	Encadreur

Promotion
2015 / 2016

Dédicaces

À mes parents…

Madoui Roumaissaa

DÉDICACES

Je dédie ce travail
À ma mère pour toute sa patience et sa persévérance durant cette période difficile.
À mon père pour son soutient moral et financier.
À mes sœurs adorables Maroua, Roufaida, Rania et chaima.
À mon cher frère Abdelhamid.
À toute la famille.
À mes collègues de l'université.
À mes enseignants.
À tous mes amis.

Ammari Naziha

Remerciement :

Nous remercions Dieu pour nous avoir donné santé, courage et patience afin de nous aider à réaliser ce travail.

Au terme de ce modeste travail nous tentons à remercier chaleureusement et respectivement tous ceux qui ont contribué de prés ou de loin à la réalisation de ce modeste projet de fin d'étude, à savoir Notre encadreuse M*me* Akli Isma et tout l'équipe de CDTA pour nous avoir accueillis.

Nous tentons aussi à remercier particulièrement notre promotrice M*me* Oukid Saliha de nous avoir suivis et guidés tout au long de réalisation de notre travail.

ملخص:

الغرض من هذا العمل هو تطوير جهاز محاكاة 3D لنظام متعدد الروبوتات الغير المتجانسة. المهمة هي إنشاء نموذج ثلاثي الأبعاد للذراع الآلية المتحركة RobuTER/ULM والبيئة التي سوف يتم فيها تطوير الروبوتات الافتراضية, ثم محاكاة سلوك أجهزة الاستشعار, لجعل المحاكاة أكثر واقعية وأخيرا توليد حركة الروبوتات.

نتائج المحاكاة تستدعي التحقق من سيناريوهات مختلفة لمحاكاة الروبوت (RobuTER/ULM) و مجموعة من الروبوتات.

Résumé :

Le but de ce travail est de développer un Simulateur 3D pour un système Multi-robots hétérogènes. Le travail consiste à créer le modèle 3D du manipulateur mobile RobuTER/ULM ainsi que celui de l'environnement dans lequel doit évoluer les robots virtuels, ensuite simuler le comportement de capteurs, afin de rendre la simulation plus réaliste. Et enfin, générer les mouvements des robots.

Les résultats de simulation font appel à des différents scenarios de validation, pour la simulation d'un robot (RobuTER/ULM) et d'un groupe des robots.

Mots clés : Robot, Système multi-robots, Simulation, Simulateur, ROS, Manipulateurs mobiles, RobuTER/ULM, Contrôleur.

Abstract:

The purpose of this work is to develop a 3D simulator for heterogeneous multi-robots system.

The aim of this work is to create the 3D model of the mobile manipulator robuTER/ULM and the environment in which evolve the virtual robots, and simulate the sensors behavior in order to make the simulation more realistic. Finally, generate the robots movements.

The simulation results call for different scenarios of validation for the simulation of a robot (RobuTER/ULM) and a group of robots.

Keywords: Robot, Multi-robot system, Simulation, Simulator, ROS. mobile manipulators. RobuTER/ULM, Controller.

Liste des tableaux

Liste des Figures

Table des matières

Introduction Générale

Ces dernières années le domaine de la robotique a considérablement augmenté, et il couvre un grand nombre de recherche, des résultats de recherche ont donné naissances à des systèmes multi-robots qui ont remplacé le système mono-robot afin de profiter de leurs avantages qui sont multiples, nous pouvons citer la fiabilité, la robustesse, la rapidité et la flexibilité.

Actuellement les systèmes multi-robots sont utilisés pour aider ou remplacer l'homme dans la réalisation de certaines tâches dans plusieurs domaines; tels que : la médecine, le domaine militaire, l'industrie etc..... . Le coût de ce type de recherche est prohibitif. Les robots sont chers, et produire de grandes quantités d'entre eux est hors de la portée du budget de la recherche d'aujourd'hui. Heureusement, des programmes de simulation multi-robots évolutifs sont venus à la rescousse.

La thématique générale abordée dans ce projet est le développement d'un Simulateur 3D pour système Multi-robots hétérogènes. Plus précisément l'objet de notre travail vise à :

- Construite le modèle 3D du robot Robuter/ULM de la division productique et robotique (DPR) du centre de développement des technologies avancées (CDTA) ainsi que celui de l'environnement dans lequel doit évoluer les robots virtuels.
- Simuler le comportement de capteurs. afin de rendre le simulateur plus réaliste.
- Générer les mouvements des robots.

Pour assurer une meilleure présentation du travail effectué et garantir la clarté du mémoire, outre cette introduction générale, ce manuscrit se compose de cinq chapitres, une conclusion générale. Chacun met en évidence une contribution particulière du travail :

- Dans le premier chapitre de ce mémoire, nous présentons un état de l'art des robots et des systèmes multi-robot, leurs domaines d'application, ainsi que leurs principales caractéristiques.
- Dans le second chapitre, nous décrivons les différents simulateurs de système multi-robot existants et ROS (Robot Operating System), ainsi que les types de simulation.
- Dans le troisième chapitre, nous proposons notre solution. Une présentation du projet et du robot d'étude, en occurrence le manipulateur mobile RobuTER/ULM, est présenté en premier lieu. En second lieu, nous décrivons les détails de la modélisation, la spécification en ayant recours aux diagrammes UML.

- Dans le quatrième chapitre, nous partons de la spécification et des diagrammes créés au cours de la conception pour implémenter la solution.
- Dans le cinquième chapitre, nous testons notre travail via différents scénarios de validation. Pour cela, les résultats de simulation sont présentés.
- Une conclusion générale, donnée à la fin du manuscrit, résume la contribution du travail effectué, discute les principaux résultats obtenus et fixe des perspectives envisageables d'amélioration.

Chapitre 1 :

Robotique

Et

Systèmes multi-robots

1. Introduction

La robotique est considérée aujourd'hui comme la source la plus importante d'innovations, susceptibles d'améliorer la productivité et la sécurité dans différent secteurs. Ce chapitre est consacré à la présentation de notions fondamentales des robots et des systèmes multi- robots.

2. Robot

Étymologiquement, le mot robot vient du tchèque robota qui signifie travail, il désigne un système automatique asservi à une unité de commande informatisée [BAM, 10]. Un robot est un ensemble de corps mobiles associés dont les mouvements sont commandés numériquement et synchronisés. Chaque mouvement élémentaire est un axe de commande numérique. Cependant un robot n'est qu'une machine programmable qui ne fait qu'exécuter ce que l'homme lui a appris, en mettant en œuvre et intégrant [DGT, 14] :

- Des capacités d'acquisition de données avec des capteurs à même de détecter et d'enregistrer des signaux physiques.
- Des capacités d'interprétation des données acquises permettant de produire des connaissances.
- Des capacités de décision qui, partant des données ou des connaissances, déterminent et planifient des actions. Ces actions sont destinées à réaliser des objectifs fournis le plus souvent par un être humain, mais qui peuvent aussi être déterminés par le robot lui-même, éventuellement en réaction à des événements.
- Des capacités d'exécution d'actions dans le monde physique à travers des actionneurs, ou à travers des interfaces.

Le robot peut également présenter [DGT, 14] :

- Des capacités de communication et d'interaction avec des opérateurs ou des utilisateurs humains, avec d'autres robots ou des ressources via un réseau comme l'internet.
- Une capacité transversale d'apprentissage, qui permet au robot de modifier son fonctionnement à partir de son expérience passée.

2.1. Catégorie de robots

Il existe différent catégories de robot :

- **Robots mobiles :** Les robots mobiles ont une place particulière en robotique. Leur intérêt réside dans leur mobilité qui ouvre des applications dans de nombreux

domaines. Ils sont destinés à assister l'homme dans les tâches pénibles (transport de charges lourdes), monotones ou dans un environnement (nucléaire, marine, spatiale, lutte contre l'incendie, surveillance…). L'autonomie du robot mobile est une faculté qui lui permet de s'adapter ou de prendre une décision dans le but de réaliser une tâche malgré un manque d'informations préliminaires ou éventuellement erronées. Dans d'autres cas d'utilisation, comme celui des véhicules d'exploration de planètes, l'autonomie est un point fondamental puisque la télécommande est alors impossible par le fait de la durée du temps de transmission des informations [KAM, 14].

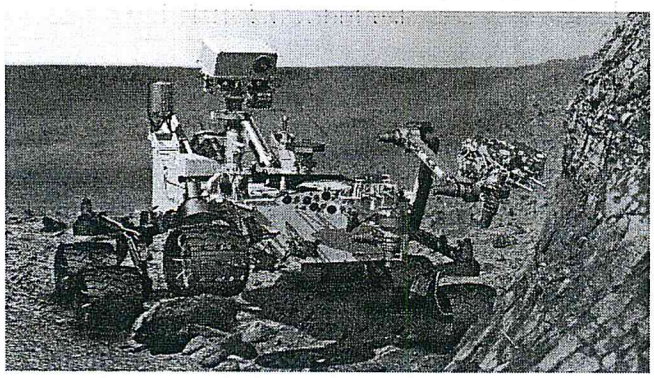


Figure 1 : *Robot mobile.*

- **Robots humanoïdes :** Catégorie la plus connue, en grande partie grâce à leur promotion faite par la science fiction, elle regroupe tous les robots anthropomorphes, ceux dont la forme rappelle la morphologie humaine. Ces robots ont généralement un torse, une tête, deux bras et deux jambes. Parfois, certains de ces robots ne représentent qu'une partie du corps. Lorsqu'un robot anthropomorphe imite non seulement l'apparence physique, mais aussi les comportements humains, on l'appelle un androïde [TJL, 13].

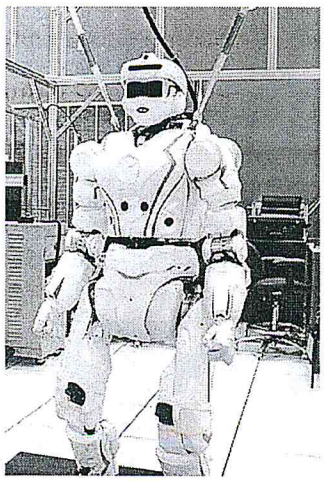


Figure 2 : *Robot humanoïde.*

- **Robots manipulateurs :** ce sont des robots à base fixe permettant de manipuler des objets. Les liens de ce manipulateur sont reliés par des axes permettant, soit du mouvement de rotation ou de translation (linéaire) de déplacement. Ils peuvent être utilisés pour l'assemblage, la manipulation d'objets, l'entretien, la réparation et pour d'autres applications [GUI, 14].

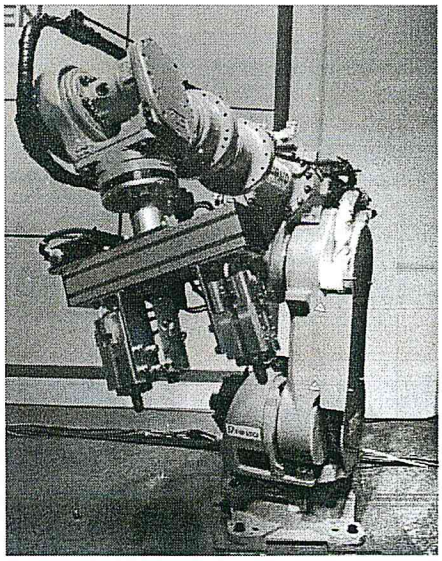


Figure 3 : *Robot manipulateur.*

2.2. Classification de robots [PAR, 08]

Il existe deux classifications :

Selon le type de tâches : il existe deux classifications de robots :

- **Robots de substitution** : pour les tâches répétitives et précisément définies (donc programmables)
- **Robots de coopération** ou **« cobot »** : les mouvements sont pilotés directement par l'homme, le robot apportant l'assistance à l'effort, à la précision du geste…

Selon le type d'architecture : il existe deux classifications:

- **Robots à structure sérielle** : SCARA, cylindriques, sphériques, cartésiens, anthropomorphes… Les exemples d'applications sont Packaging/Conditionnement, Peinture/Pulvérisation, Soudage avec procès intégré, Vissage, Palettisation, Positionnement, Chargement, Polissage...
- **Robots à structure parallèle** (6 degrés de liberté) : par exemple Robots Delta. Exemples d'applications : Assemblage / Manutention / Encaissage, Prise et dépose à la volée « Pick and place »

3. Système multi robots [IME, 13]

Un système multi robots est un système constitué d'un ensemble de robots, homogènes ou hétérogènes, mobiles ou fixes. Ces robots sont capables de communiquer entre eux et coopérer pour améliorer l'efficacité d'exécution des tâches ou encore pour permettre d'exécuter des tâches impossibles à exécuter de façon individuelle. Un système multi robots peut effectuer des tâches difficiles ou même impossibles à accomplir par un seul robot. Une équipe de robots fournit une certaine redondance, elle contribue à l'accomplissement d'une tâche de manière collaborative. Ce système offre plusieurs avantages par rapport aux systèmes mono-robot :

- La complexité de la tâche à réaliser est trop élevée pour un seul robot donc elle nécessite une coopération d'un ensemble de robots.
- La construction de plusieurs robots simples avec des ressources bornées est beaucoup plus facile que la construction d'un seul robot complexe et puissant.
- L'introduction de plusieurs robots augmente la robustesse grâce à la redondance.
- La rapidité : pour obtenir un niveau de performance élevé les tâches sont effectuées en parallèle. A titre d'exemple : l'exploration parallèle d'un environnement inconnu par un ensemble de robots mobiles, dans le but de faire soit une cartographie de l'environnement, soit la réalisation d'une tâche de fourragement. Ces deux tâches peuvent être réalisées par un seul robot, mais l'ajout d'autres robots va faire en sorte d'accélérer l'exécution des tâches en question.
- La robustesse : les performances du contrôle dans un SMR ne sont pas trop affectées en cas de défaillance d'un robot.
- La flexibilité : il est possible d'exécuter les tâches désirées, de diverses manières. Ceci est induit principalement par la redondance des entités robotiques.

3.1. Types de système multi-robots

Il existe deux types de système multi-robot [ZIN, 15] :

- **Système multi robots homogènes :** est un ensemble de robots qui ont les mêmes caractéristiques, ils doivent coopérer entre eux, ils s'auto-organisent pour soulever, porter pousser et déposer l'objet qu'un seul robot ne pourrait réaliser tout seul.
- **Systèmes multi robots hétérogènes :** est un ensemble de robots mixtes, c'est-à-dire les robots ne sont pas identiques, il peut y avoir des robots fixes et des robots mobiles, dans ce cas chaque robot s'occupe de son propre sous tâche, les robots manipulateurs

fixes s'occupent du dépôt et du soulèvement d'objet et les robots mobiles s'occupent de son transport.

3.2. Domaines d'utilisation de système multi-robots :

L'un des objectifs de l'utilisation des robots est de remplacer ou d'aider les êtres humains dans la réalisation des certaines tâches. C'est particulièrement souhaitable pour des tâches dangereuses comme l'exploitation spatiale et sous-marine, l'exploration de l'intérieur d'une centrale nucléaire ou encore fastidieux tel que la surveillance d'un site, les applications militaires. Les robots peuvent effectuer des tâches automatiquement, mais ils sont aussi dotés d'une certaine intelligence [TUA, 10]. Nous citons dans ce qui suit quelques exemples de domaines d'utilisation de système multi-robots :

- **Robotique médicale :**

La robotique se développe et se perfectionne au grand bénéfice de la médecine. Un robot médical est un robot qui peut soit être piloter par un médecin et cela en vu d'une meilleure précision ou alors exécuter des opérations délicate qui auront ainsi plus de chance de réussir. De plus les robots ont la capacité d'intervenir sur des cellules cible ce qui a pour effet de limiter les effets secondaires après une opération ou après l'injection de médicament, en général c'est un appareil aide à soigner des patients [FOU, 13].

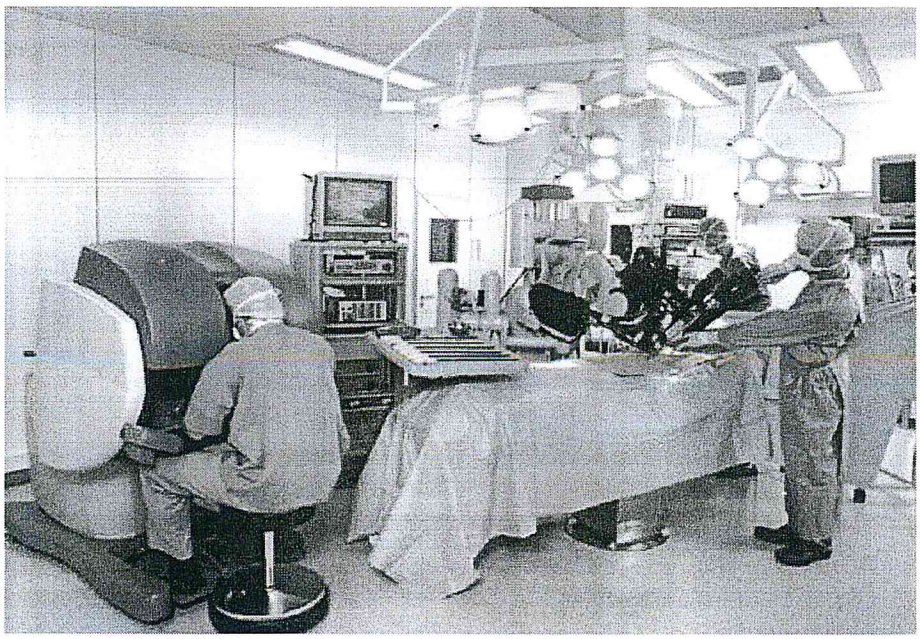


Figure 4 : *Robots au bloc opératoire.*

- **Robotique industrielle :**

Les robots industriels ont été développés pour intervenir dans les milieux dangereux (nucléaire), ils servent aussi dans le maniement d'objets lourds, des robots de peinture et de soudure dans l'industrie automobile, ils sont recommandés pour des tâches répétitives et précises [HAD, 11].

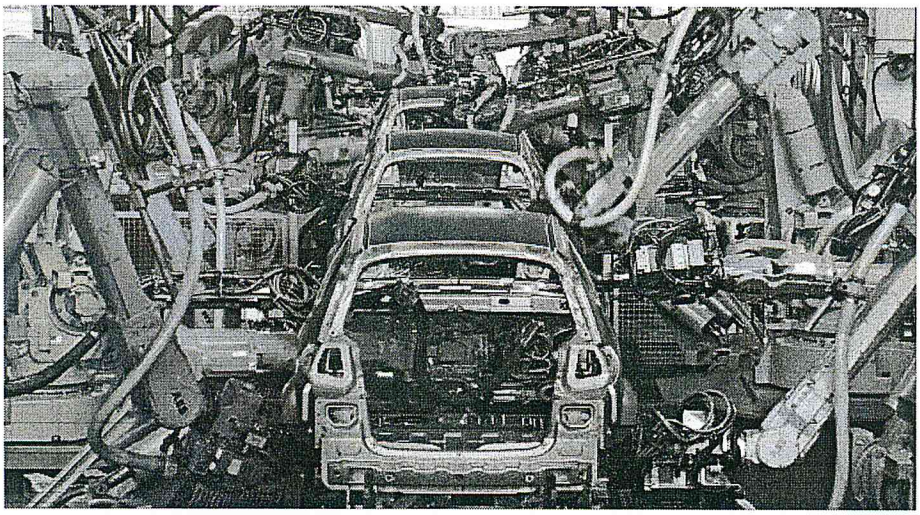


Figure 5 : *Robots industrielle.*

- **Robotique militaire :**

Un robot militaire est un robot autonome ou contrôlé à distance conçu pour des applications militaires, les robots sont plus précis dans leurs tirs et aussi beaucoup plus dure à détruire qu'un soldat. De plus il y a des avantages d'un point de vue économique car les robots n'ont besoin d'aucune formation puisque ils fonctionnent à partir de logicielles, et ils n'ont pas besoin de nourriture ou de soins médicaux. Cependant les prix des robots de guerre sont excessivement élevés [WEE, 06].

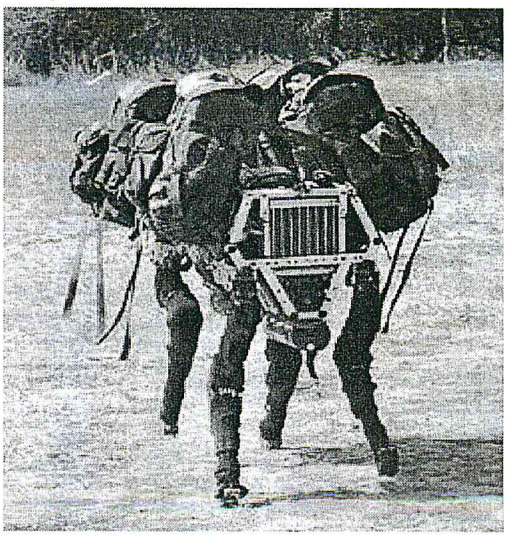


Figure 6 : *Robot militaire.*

4. Conclusion :

Dans ce chapitre, nous avons présenté de façon globale les définitions et explications sur les concepts de robot et de Système Multi robots et une vision d'ensemble des caractéristiques et techniques associées aux systèmes multi-robots. Le prochain chapitre sera consacré à la simulation des systèmes multi-robots.

Chapitre2 :

Simulation des systèmes multi-robots

1. Introduction :

Dans le contexte de la robotique mobile, la simulation est aujourd'hui un outil incontournable pour L'étude et la conception de systèmes multi-robots. En supprimant les difficultés matérielles liées aux expérimentations réelles, elle permet de focaliser l'attention sur les aspects comportementaux et organisationnels du système. Nous présentons ici des outils de simulations dédiés aux systèmes multi-robot.

2. Les simulateurs :

Un logiciel de simulation de robot ou un simulateur est un environnement virtuel complet utilisé pour créer des applications, des robots simples qui peuvent être construits, programmés et testés. Dans quelque cas, ces applications peuvent être transférées sur le robot réel sans modification, ce qui réduit les coûts et le temps [ALM, 14]. Il existe plusieurs simulateurs dédiés aux systèmes multi-robot. Nous présentons dans cette partie quelques simulateurs :

2.1. Simulateurs commerciaux:

Considérons d'abord les simulateurs commerciaux :

- **V-REP :** est un simulateur 3D compatible avec Windows, Mac et Linux, qui permet la modélisation d'un système entier ou seulement certains composants comme les capteurs, les mécanismes, les engrenages, etc. Les scripts sont connectés directement aux véritables plateformes robotiques s'ils sont utilisés avec ROS. V-REP peut être utilisée pour contrôler la partie matérielle, développer des algorithmes, créer des simulations d'automatisation d'usine, ou pour démonstration éducatives [NOG, 14]. Le logiciel se base sur le langage C/C++, Python, Java, Urbi, Matlab/Octave et fonctionne sous Linux, MacOS X, Windows [IVF, 14].
- **Webots :** est un environnement de développement utilisé pour modéliser, programmer et simuler des robots mobiles. Avec Webots, l'utilisateur peut concevoir des configurations robotiques complexes, avec un ou plusieurs robots, semblables ou différents, dans un environnement partagé. L'environnement peut être modifié à volonté par les utilisateurs qui peuvent insérer et éditer des objets de poids et de formes différents dans l'environnement simulé. Les programmes de contrôle du robot peuvent être écrits en C, C++, Java, Python et MATLAB [GON, 13].
- **XDE :** est basé sur un noyau de simulation de la physique qui traite les corps rigides et déformables, en particulier des câbles, des systèmes multi corps avec contraintes

cinématiques et des contacts intermittents. XDE offre un large panel de moyens d'interaction, offrant la perception et la manipulation des capacités de l'utilisateur sur chacun des éléments de simulation de vision 3D stéréoscopique colocalisés avec la capture de mouvement, par capture de mouvement, réaliste feed-back sonore, métaphores visuelles, couplage avec différents systèmes de capture de mouvement et couplage avec des dispositifs tactiles. Il est basé sur des technologies largement acceptées comme Python, C ++, Ogre, OpenMP, CUDA. L'environnement est disponible sur les deux plates-formes Microsoft Windows et Linux, pour les deux architectures 32 bits et 64 bits [POB, 14].

- **ROBOTRAN :** est un logiciel qui génère des modèles symboliques de systèmes multi-corps. Il est développé par le Centre de Recherche en mécatronique de l'Université Catholique de Louvain. En outre, l'environnement de simulation est proposé dans Matlab et Simulink. Cela permet de tirer parti de l'interface et des boîtes à outils Matlab direct à traiter les données et analyser les résultats. Le code source est écrit en C / C ++, il est disponible dans multiplateforme (Linux, Windows et Mac OS) [IVF, 14].
- **Easy-Rob :** est un simulateur 3D pour la planification et la simulation dans les usines qui opèrent au sein des cellules de travail. Easy-Rob permet à l'utilisateur de programmer et visualiser des processus 3D à l'aide de robots (robots simples ou multiples) dans les tâches industrielles telles que la manutention, l'assemblage, le revêtement et l'étanchéité. Le code des contrôleurs de simulation ne peut pas être mis en œuvre pour des robots réels. Il peut simuler plusieurs robots et la cinématique synchronisés en temps réel. Il est disponible uniquement dans la plateforme Windows (Windows 7, XP and vista) pour les deux architectures 32 bits et 64 bits. Il fonctionne avec la librairie graphique OpenGL [GON, 13].

2.2. Simulateurs Open-source:

Dans cette partie nous allons présenter les simulateurs open source les plus connus :

- **Morse (Multi open robots simulator engine) :** permet d'évaluer l'intégration des robots dans un environnement complexe. Grâce à Blender, un outil de modélisation 3D en temps réel, la plate-forme permet d'analyser le comportement d'un ou plusieurs robots. Même leurs interactions avec les objets ou les humains sont prises en compte. Morse permet de simuler des robots dans un environnement réaliste et intègre plusieurs fonctionnalités, comme la possibilité d'ajouter des éléments pour les besoins

de la simulation. Il prend ainsi en charge vingt types de capteurs (caméra, scanners…) et quinze types d'actionneurs. Le logiciel se base sur le langage Python et fonctionne sur Linux, et avec certaines limitations sur OSX et Windows [DAV, 14].

- **Gazebo :** est un simulateur multi-robots pour les environnements extérieurs et intérieurs. Il prend en charge des simulations pour odométrie et des capteurs robotiques standard comme sonar, laser, GPS, caméra ..., et toutes sortes de modèles de robots qui sont similaires à PR2, Pioneer2DX, Pioneer 2AT. Il est capable de simuler des environnements très réalistes. En outre, il permet également de simuler la dynamique des corps rigides de façon très précise, ce qui nécessite d'ailleurs une bonne carte graphique et processeur CPU et OpenGL Lib. Par conséquent, le nombre de robots simulés en temps réel se limite à la performance du matériel. Gazebo a été commencé comme un projet à l'Université de Californie du Sud. Par la suite, il a été intégré dans le cadre ROS par John Hsu, qui était un chercheur à Willow Garage. Gazebo compatible avec ROS. En conséquence, un client écrit pour Gazebo peut généralement fonctionner sur ROS avec peu ou pas de modification. Il fonctionne sur linux seulement. Les contrôleurs peuvent être écrits en C, C ++, Python et Java en utilisant Player ou ROS et le code écrit est portable à des véritables plates-formes robotiques [NOG, 14] [IVF, 14].
- **RDS (Robotics Developer Studio) :** est produit par Microsoft depuis quelques années et est devenu gratuit en mai 2010. Le langage de développement visuel VPL (Visual Programming Language) permet de créer des applications par glisser-déposer de composants prédéfinis. Un runtime, CCR (Concurrency and Coordination Runtime), gère en mode asynchrone les échanges entre le matériel, robot, capteurs, etc. et le logiciel qui le commande. L'environnement de simulation VSE (Visual Simulation Environment) permet d'expérimenter dans un monde virtuel en 3D l'interaction du robot avec des objets [KHU, 11].
- **Argos :** est conçu pour simuler des expériences complexes impliquant de grands essaims de robots de différents types. Argos est le premier simulateur multi-robot qui est à la fois efficace (rapide avec de nombreux robots) et flexible (extrêmement personnalisable pour des expériences spécifiques). Dans Argos, il est possible de partitionner l'espace simulé en plusieurs sous-espaces, gérés par différents moteurs physiques fonctionnant en parallèle. Deuxièmement, l'architecture Argos est multi-thread, donc conçu pour optimiser l'utilisation de processeurs multi-core modernes.

Enfin, l'architecture d'Argos est très modulaire, ce qui permet d'ajouter facilement des fonctionnalités personnalisées et allocation appropriée des ressources de calcul. Les contrôleurs de robots sont écrits en C ++ [BMF, 12].

- **MRSim (Multi-Robot Simulator) :** est une extension du simulateur de robot mobile autonome SIMROBOT (SIMulated ROBOTs). MRSim permet à l'utilisateur de simuler le comportement de plusieurs robots mobiles dans les environnements virtuels. Par rapport à son prédécesseur SIMROBOT, il est équipé des nouvelles versions de MatLab. Il est également équipé pour répondre aux applications multi-robots. Bien que SIMROBOT a été doté de plusieurs fonctionnalités intéressantes pour la robotique mobile, il a présenté plusieurs limitations pour les applications multi-robots. Par conséquent, MRSim a été créé principalement pour permettre aux utilisateurs de développer des applications multi-robots, qui bénéficieraient de travailler avec certaines exigences spécifiques telles que la communication multi-hop. En outre, la plupart des fonctionnalités dans MRSim ont une aide intégrée (qui peut être consultée en tapant simplement la fonction d'aide) qui permet de comprendre facilement comment créer et executer des simulations. En résumé, tout comme SIMROBOT, chaque robot dans MRSim peut être équipé de plusieurs capteurs virtuels et peut être entraîné par son propre algorithme de commande. La boîte à outils comprend deux applications indépendantes. Le premier est l'éditeur (SimEdit), qui permet à l'utilisateur de créer et modifier l'environnement virtuel, modifier les algorithmes de contrôle de robots, charger et enregistrer la simulation, et autres. La seconde application Simulator (simview), peut être exécutée directement à partir de l'éditeur ou séparément de la fenêtre de commande MatLab et sert en tant que spectateur de simulation [COU, 12].

3. Comparaison entre les simulateurs

Les tableaux ci dessous comparent les simulateurs et leurs caractéristiques. Tableau 1 compare les simulateurs commerciaux tandis que Tableau 2 compare les simulateurs open-source.

	Morse	**Gazebo**	**RDS**	**Argos**	**MRSim**
Principal langage de Programmation	Python	C++	VPL	C++	Matlab
système d'exploitation	Linux BSD* MacOS X	Linux, MacOS X, Windows	Windows	Linux, MacOS X	Linux, MacOS X, Windows
Interface utilisateur Graphique	Non	Oui	Oui	Oui	Non
Réalisme de l'environnement	Oui	Oui	Oui	Non	Non
Capteurs	Odometry, range, camera...	Odometry, range, camera	Odometry, range, camera	Odometry, range, camera	Odometry, range, camera

Tableau 1 *: Comparaison des simulateurs open-source.*

	V-REP	**Webots**	**XDE**	**Robotran**	**Easy-rob**
Principal langage de Programmation	LUA	C--	Python	C++	C++
système d'exploitation	Linux MacOS X Windows	Linux MacOS X Windows	Linux Windows	Linux MacOS X Windows	Windows
Interface utilisateur Graphique	Oui	Oui	Oui	Oui	Oui
Réalisme de l'environnement	Oui	Oui	Non	Non	Non
Capteurs	Camara, range	Odometry, range, camera, GPS	Sonore, métaphores visuelles,	Odometry	Odometry

Tableau 2 : *Comparaison des simulateurs commerciaux.*

Comme nous avons indiqué dans la section 2, beaucoup de bons Simulateurs Sont disponibles. Tenant compte du réalisme de l'environnement, le réseau de capteurs disponibles et le langage de programmation python. Gazebo à été choisis pour simuler les différents modèles du monde réel et les robots. L'utilisation de Gazebo nécessite l'installation du ROS qui permet de piloter la simulation.

4. ROS (Robot Operating System)

ROS (Robot Operating System) est une plateforme de développement logicielle pour robots. Il s'agit d'un méta-système d'exploitation qui peut fonctionner sur un ou plusieurs ordinateurs et qui fournit plusieurs fonctionnalités : un simulateur 3D (Gazebo), l'abstraction du matériel, le contrôle des périphériques de bas niveau, la mise en œuvre de fonctionnalités

couramment utilisées, la transmission de messages entre les processus et la gestions des packages installés [LAP, 11]. ROS peut fonctionner sur un ou plusieurs ordinateurs, il a été développé en 2007 par la société américaine Willow Garage pour son robot PR2 (Personal Robot 2). Son développement est aujourd'hui mené par l'Open Source Robotics Foundation (OSRF). ROS est officiellement supporté par plus de 75 robots, il se programme en C++ et python. Le logiciel ROS peut être séparé en trois groupes [WIK, 16] :

- **Outils pour créer, lancer et distribuer des logiciels basés sur ROS :**
  - **Roscore :** est une collection de noeuds et de programmes qui constituant des conditions préalables d'un système à base de ROS. Il est nécessaire d'avoir un « roscore » en cours d'exécution pour que les nœuds ROS puissent communiquent.
  - **Roslaunch :** lit un fichier xml qui contient tous les paramètres pour lancer une application distribuée ROS. Il constitue un moyen pratique pour démarrer plusieurs nœuds, ainsi que d'autres exigences d'initialisation telles que le réglage des paramètres.
  - **Catkin :** système de compilation et de gestion de *package*s. Catkin fournit le concept d'espaces de travail, où nous pouvons construire plusieurs packages interdépendants à la fois.
- **Client ROS :** est une collection de codes qui facilite le travail du programmeur ROS. il prend beaucoup de concepts ROS et les rend accessibles via le code. En général, ces bibliothèques permettent d'écrire des nœuds ROS, publier et abonner aux topics, écrire et appeler des services, et utiliser le Parameter Server. Une telle bibliothèque peut être mise en œuvre dans n'importe quel langage de programmation. Il existe deux Bibliothèques clientes principales : roscpp (C++) et rospy (python).
- **package :** c'est l'unité principale d'organisation logicielle de ROS. Un package est un répertoire qui contient les nœuds, les librairies externes, des données, des programmes pour ROS utilisant un ou plusieurs clients ROS, des fichiers de configuration et un fichier de configuration xml nommé manifest.xml.

4.1. Niveau du système de fichiers ROS [LEN, 15]

Semblable à un système d'exploitation, les fichiers ROS sont également organisés sur le disque dur d'une façon particulière. Le graphique ci-dessous montre comment les fichiers ROS et les dossiers sont organisés sur le disque:

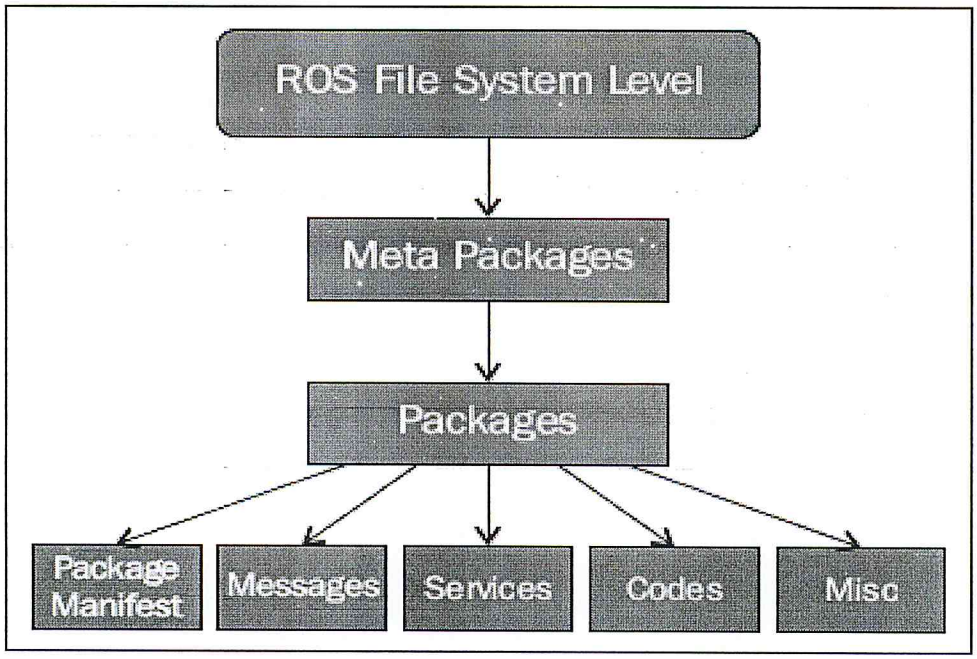


Figure 7 : *Niveau du système de fichiers ROS.*

a) Meta packages :

Le Meta package regroupe un ensemble de plusieurs packages comme un ensemble logique unique.

b) Package :

Une structure typique d'un Package ROS est montrée sur la figure suivante :

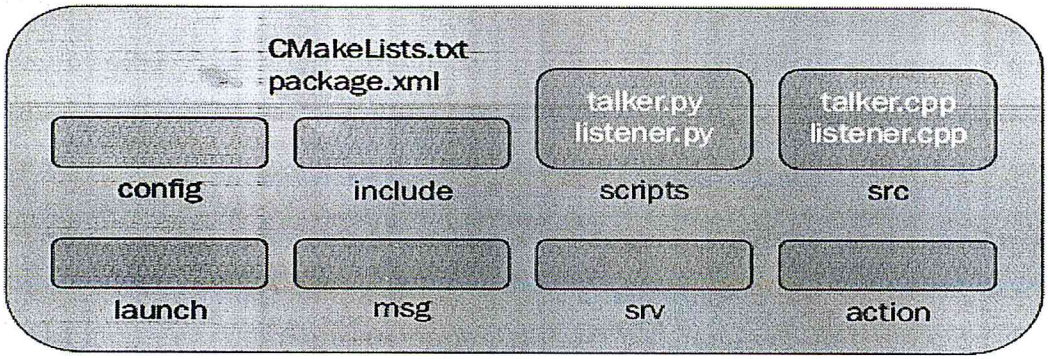


Figure 8 : *Une structure typique d'un package ROS.*

- **Config :** tous les fichiers de configuration qui sont utilisés dans un package ROS sont conservés dans ce dossier. Ce dossier est créé par l'utilisateur. Il est une pratique courante de nommez le dossier config pour conserver les fichiers de configuration.
- **Include/nom_du_package :** ce dossier se compose d'en-têtes et des bibliothèques qui peuvent être utilisée à l'intérieur du package.
- **Scripts :** ce dossier conserve les scripts exécutables Python ou C++.
- **Src :** ce dossier stocke les codes sources C++ ou Python.

- **Launch :** ce dossier conserve les fichiers de lancement qui sont utilisés pour lancer une ou plusieurs nœuds ROS.
- **Msg :** ce dossier contient des définitions de messages personnalisés qui sont un type d'information qui est envoyé d'un processus ROS à l'autre.
- **Srv :** ce dossier contient les définitions de service qui est une sorte d'interaction (demande /réponse) entre les processus.
- **Action :** ce dossier contient la définition d'actions.
- **package.xml :** contient des informations sur le package, auteur, licence, dépendances, compilation, version, gestionnaire, etc.
- **CMakeLists.txt :** est la CMake qui permet de construire le package ROS.

4.2. Architecture de communication

ROS offre une architecture souple de communication interprocessus et inter-machine [WIK, 16].

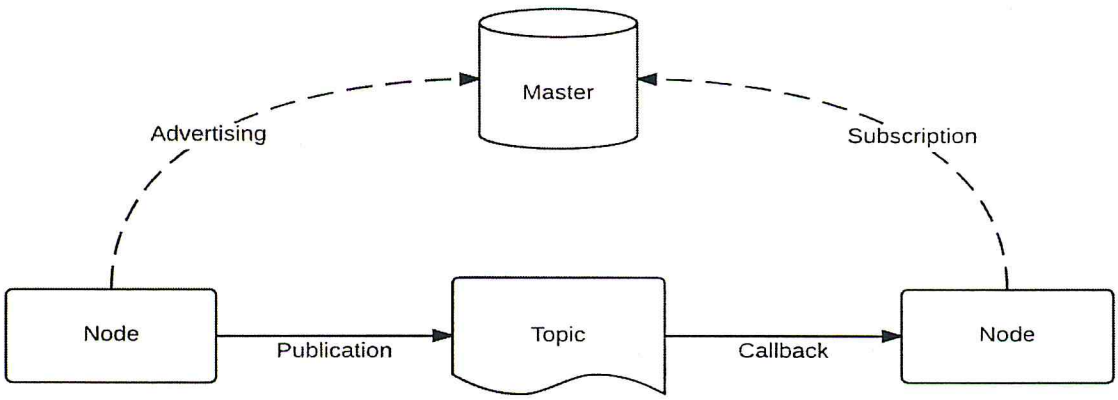


Figure 9 : *Schéma de fonctionnement du Master sous ROS.*

- **Nœud :** les processus ROS sont appelés des nœuds, est une instance d'un exécutable. Un nœud peut correspondre à un capteur, un moteur, un algorithme de traitement, de surveillance… Chaque nœud ROS est écrit en utilisant les bibliothèques clientes ROS telles que Rospy et Rosccp [LEN, 15].
- **Master :** est un service de déclaration et d'enregistrement des nœuds qui permet ainsi à des nœuds de se connaitre et d'échanger de l'information. Le Master est implémenté via XMLRPC. Le Master comprend une sous-partie très utilisée qui est le Parameter Server. Celui-ci, également implémenté sous forme de XMLRPC. Le Parameter Server est une sorte de base de données centralisée dans laquelle les nœuds peuvent stocker de l'information et ainsi partager des paramètres globaux [LEN, 15].

- **Topic :** est un système de transport d'information basé sur le système d'abonnement / publication (subscribe/publish). Un ou plusieurs nœuds pourront publier de l'information sur un topic et un ou plusieurs nœuds pourront lire l'information sur ce topic [LEN, 15].

Chaque nœud peut communiquer avec d'autres *via* des topics. La connexion entre les nœuds est gérée par un Master et suit le processus suivant [WIK, 16] :

- Un premier nœud averti le Master qu'il a une donnée à partager.
- Un deuxième nœud averti le Master qu'il souhaite avoir accès à une donnée.
- Une connexion entre les deux nœuds est créée.
- Le premier nœud peut envoyer des données au second.

5. Description du robot dans Gazebo et ROS

Un robot est un ensemble de corps dont les mouvements relatifs sont contraints. La modélisation des contraintes passe par la modélisation des articulations reliant deux corps entre eux. Le format URDF est utilisé dans ROS pour la simulation et les tests. Il est un type de fichier pris en charge pour gazebo et d'autres outils ROS. L'URDF définit deux grands types d'objets : les corps et les joints [MOU, 12].

a) Corps :

Un corps est défini par une forme géométrique. Cette forme peut soit être définie par une primitive géométrique comme une sphère, une cylindre, un cube, ou bien via un modèle 3D. Les informations de la dynamique du corps sont également codées : position du centre de masse au sein de l'objet ainsi que la matrice d'inertie associée au corps. Une représentation alternative de la géométrie du corps utilisée pour les calculs des collisions peut également être définie.

b) Articulation :

Une articulation lie deux corps tout en imposant un ensemble de contraintes au mouvement relatif de ces deux éléments. Une articulation comporte donc un lien vers le corps "père" et le corps "fils" pour définir sa position dans l'arbre cinématique. Dans le cas d'articulation racine, il peut ne pas y avoir de corps "père" et dans le cas d'un organe terminal, il peut ne pas y avoir de corps "fils". Les articulations à des degrés de liberté qui possèdent généralement un minimum et maximum qui sont soit le résultat de la conception mécanique d'articulation, soit lié à la géométrie du robot. Chaque articulation comporte donc ces valeurs ainsi qu'une vitesse et une force ou un couple maximum selon le type d'articulation. Enfin la friction

statique d'articulation ainsi que l'amortissement utilisé pour la simulation de ce dernier peuvent également être enregistrés. Les articulations supportés sont les suivants [COL, 13] :

- **Revolute :** une articulation qui pivote sur un axe et possède une gamme limitée spécifiée par des limites supérieure et inférieure.
- **Continuous :** une articulation continue qui tourne autour d'un axe et elle n'a pas de limites supérieure et inférieure.
- **Prismatic :** est une articulation coulissant qui coulisse le long de l'axe, et possède une gamme limitée spécifiée par des limites supérieure et inférieure.
- **Fixed :** cette articulation ne possède pas de degré de liberté et impose un mouvement rigide entre les deux corps.
- **Floating :** cette articulation virtuel possède six degrés de liberté et n'impose aucune contrainte.
- **Planar :** cette articulation possède deux degrés de liberté et impose un mouvement coplanaire à un plan spécifique à l'articulation.

Une fois l'arbre cinématique défini et annoté par les corps du robot auquel ils sont attachés, il reste encore un élément clé à spécifier : les capteurs qui sont des dispositifs sensibles à un phénomène déterminé et transformant cette grandeur physique en signal. Les capteurs les plus couramment utilisés en robotique [FIL, 05]:

- **Les capteurs proprioceptifs :** les capteurs proprioceptifs permettent une mesure du déplacement du robot.
- **Les télémètres :** il existe différents types de télémètres, qui permettent de mesurer la distance à l'environnement, utilisant divers principes physiques. On trouve les télémètres à ultrason, les télémètres à infrarouge et les télémètres laser.
- **Les cameras :** l'utilisation d'une caméra pour percevoir l'environnement est une méthode attractive car elle semble proche des méthodes utilisées par les humains. Le traitement des données volumineuses et complexes fournies par ces capteurs reste cependant difficile à l'heure actuelle, même si cela reste une voie de recherche très explorée.
- **Autres capteurs :** il y a d'autre type de capteur comme les capteurs tactiles (ou capteur de choc), les boussoles, les balises et le GPS.

6. Types de simulation :

En générale il existe deux types de simulation : les simulations dites continues et les simulations dites discrètes. Les simulations multi robots sont des simulations discrètes. Elles se décomposent en deux grandes classes [ERD, 96] :

- **les simulations discrètes par pas de temps :** les pas de temps dans une simulation discrète se définissent comme des intervalles de temps réguliers qui sont parcourus au cours de la simulation. Le pas de temps peut prendre différentes unités (secondes, minutes, jours...). Lors de l'incrémentation d'un pas de temps, tous les robots de la simulation sont mis à jour et/ou exécutés. En d'autres termes, nous incrémentons toute la simulation d'une unité de temps.
- **les simulations discrètes par événements :** utilisent des listes ordonnées d'événements, et traitent donc le premier événement qui est chronologiquement dans la liste.

Le type de simulation est généralement déterminé par le système que nous souhaitons modéliser. Dans notre cas, il convient d'utiliser la simulation par pas de temps afin d'observer l'évolution au cours du temps pour chaque robot et pour l'ensemble de la simulation.

7. Conclusion

Dans ce chapitre, nous avons commencé par la présentation quelques simulateurs commerciaux et open-source. Ensuit nous avons fait une comparaison entre ces derniers. Après nous avons porté notre choix sur le simulateur Gazebo comme un outil de simulation dans notre travail. Et enfin nous avons parlé du ROS (Robot Operating System) et de types de simulation. Le chapitre suivant sera dédié à l'analyse et la conception de notre solution.

Chapitre 3 :

Analyse
et
Conception

1. Introduction :

Dans ce chapitre nous allons intéresser à la conception de notre travail en utilisant le langage de modélisation UML (Unified Modeling Langage). La conception a pour objectif de permettre de formaliser les étapes préliminaires du développement d'un système afin de le rendre plus fidèle aux besoins du client. Avant de commencer la modélisation, nous allons présenter notre projet et analyser les différentes solutions existantes.

2. Présentation du projet

Dans cette partie, nous allons présenter le cadre général et l'objectif de notre travail.

2.1 Cadre générale du travail:

Les outils de simulation sont essentiels en robotique. Un simulateur permet de rendre possible la validation des algorithmes, le test de faisabilité de scenarii réalistes et offrir la possibilité de simuler efficacement des populations de robots dans des environnements intérieurs ou extérieurs. Il existe au niveau de la division productique et robotique au CDTA des robots de types différents, tels que des robots mobiles, bras manipulateurs et des manipulateurs mobiles. Le sujet proposé est le développement d'un simulateur 3D permettant de simuler le comportement des robots dans un environnement intérieur.

2.2 Objectif :

Dans le cadre de ce stage il est demandé de construite le modèle 3D de robot Robuter/ULM ainsi que celui de l'environnement dans lequel doit évoluer les robots virtuels (le laboratoire Robotique de la Division Productique et Robotique du Centre de Développement des Technologies Avancées), puis simuler le comportement de capteurs, afin de rendre le simulateur plus réaliste. Enfin, générer les mouvements des robots.

3. Présentation du robot Robuter/ULM

ROBUTER/ULM est un robot mobile manipulateur, développé par la société française ROBOSOFT, il est constitué d'une plate-forme mobile sur laquelle est installé un bras manipulateur ultra léger à six degrés de liberté. Le tout est contrôlé et commandé par un PC embarqué et trois cartes électroniques de commande à base de µcMPC555. Le robot mobile ROBUTER, et une plate forme différentielle comporte deux roues commandées indépendamment fixes en avant assurant sa mobilité et deux roues foules en arrière pour maintenir l'équilibre et la stabilité du robot. Ce robot est doté d'un système sensoriel varié qui comporte plusieurs capteurs proprioceptifs et extéroceptifs. Il a un système télémètre Laser (le

LMS SICK200), une odométrie et un capteur indicateur de niveau de charge pour les batteries. Le bras manipulateur ULM est un bras ultra léger articulé à six liaisons rotoïdes (6 degrés de liberté), avec une pince en buté. Ii est équipé par des capteurs de position pour chaque articulation et d'un capteur d'effort intégré au niveau de la pince qui permet à l'opérateur de connaître les efforts exercés par le bras sur son environnement et l'inverse afin de bien contrôler le mouvement du bras. Le bras dispose aussi d'une caméra CCD monochrome embarquée sur la pince du bras et d'un système de transmission HF.

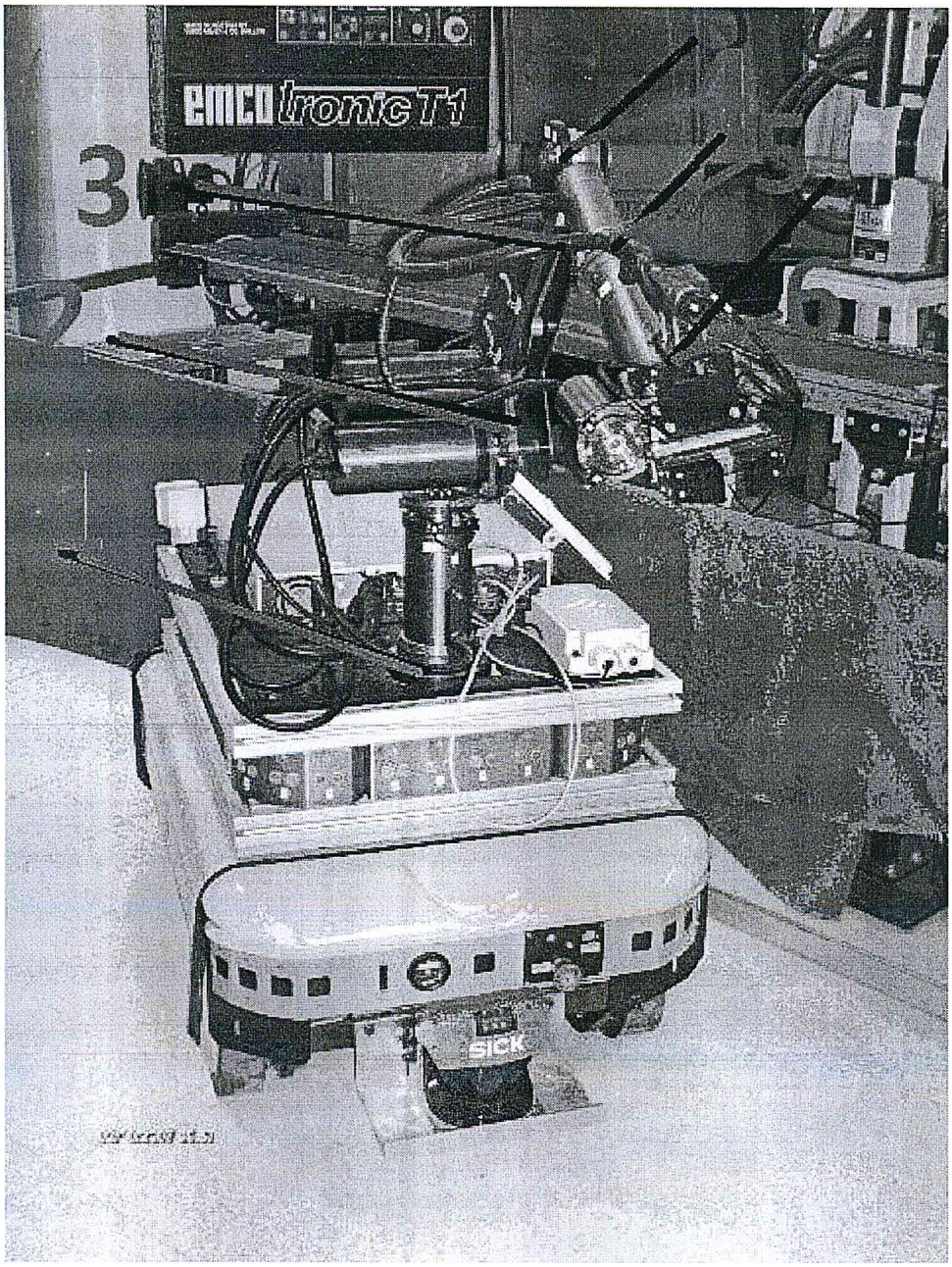


Figure 10 : *Robot mobile manipulateur Robuter/ULM.*

Le tableau suivant montre les limites de chaque articulation du bras manipulateur ULM :

Articulation	Limites d'articulation du bras	
	min(°)	max(°)
1	*-94*	*94*
2	-22	*87*
3	0	158
4	-77	77
5	-73	38
6	-55	*55*

Tableau 3 : *Limite des articulations du bras manipulateur ULM.*

4. Analyse :

Il existe un travail concernant la création du modèle 3D du robot RobuterULM au niveau de CDTA, où ils ont utilisé Solidworks (un logiciel de CAO), mais l'origine et la forme de ces pièces ne permettent pas de mettre en place un fichier URDF correct, car l'exportation du modèle nécessite un respect d'une certaine hiérarchie. À cause de ces raisons, il est nécessaire de reconstruire les pièces et les rassembler à nouveau.

5. Modélisation avec UML

UML (Unified Modeling Language) est une notation permettant de modéliser un problème de façon standard. Ce langage est né de la fusion de plusieurs méthodes existant auparavant (OMT, OOSE et BOOCH) [PIL, 15]. L'étude d'un problème est réalisée suivant trois aspects [ESP, 00]:

- **Aspect fonctionnel :** l'objectif de la capture des besoins ou exigences consiste à déterminer ce que le système informatique doit faire, c'est-à-dire le quoi afin de favoriser une meilleure compréhension des fonctionnalités du système lors du développement. Généralement, cet aspect est représenté par un diagramme de cas d'utilisation.
- **Aspect dynamique :** où on précise le comportement des objets, les différents états par lesquels ils passent et les événements qui déclenchent ces changements d'états. Cet aspect est représenté par un diagramme de séquences ou un diagramme d'activités.
- **Aspect statique :** où on identifie les propriétés des objets et leurs liaisons avec les autres objets. Cet aspect est souvent représenté par un diagramme de classes.

Dans notre projet, nous avons utilisé les diagrammes suivants :

5.1. Diagramme des cas d'utilisation :

Un diagramme de cas d'utilisation capture le comportement d'un système. Il permet d'exprimer les besoins des utilisateurs afin d'entamer la partie de développement. Ce diagramme permet de donner une vision globale du comportement fonctionnel de notre système ainsi que les acteurs (utilisateurs) et leurs interactions avec ce système [AUD, 06]. Dans ce qui suit, nous présentons les différents diagrammes de cas d'utilisation de notre système :

5.1.1. Diagramme de cas d'utilisation générale :

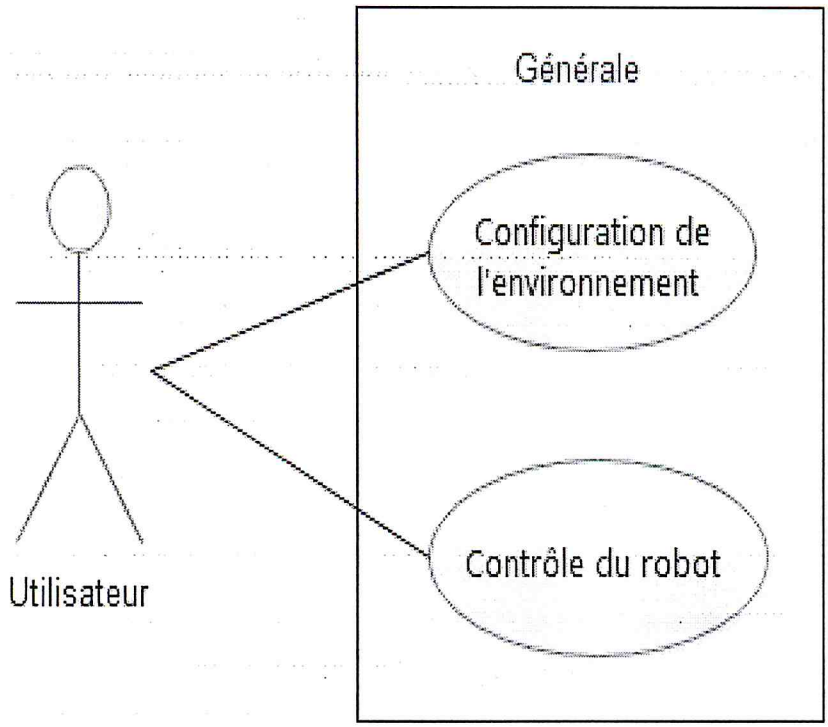


Figure 11 : *Diagramme de cas d'utilisation Générale.*

Cas d'utilisation	Acteurs	Description
Configuration de l'environnement	Utilisateur	- Lancer le modèle d'environnement et du robot. - Ajouter un objet. - Supprimer un robot ou un objet dans l'environnement.
Contrôle du robot	Utilisateur	- Contrôler le modèle robot par un programme python afin d'accomplir une tâche.

Tableau 4 : *Description du Diagramme de cas d'utilisation Générale.*

5.1.2. Diagramme de cas d'utilisation Configuration de l'environnement :

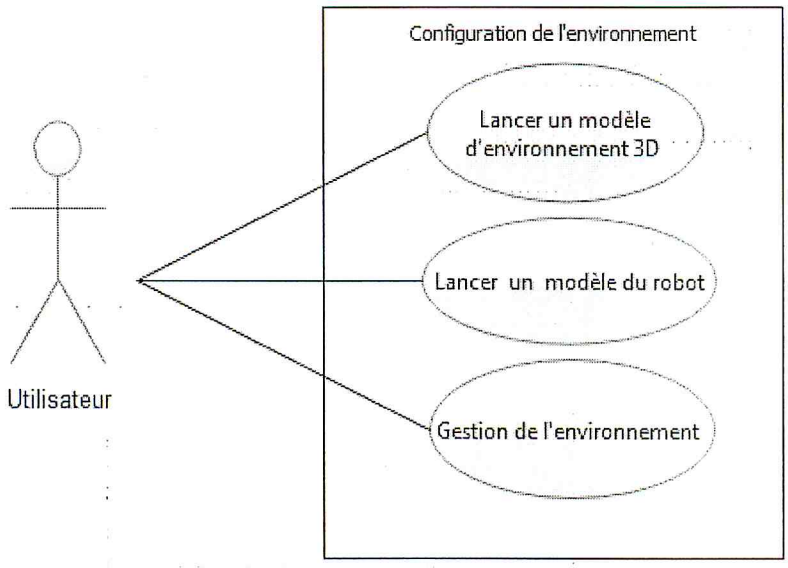


Figure 12 : *Diagramme de cas d'utilisation Configuration de l'environnement.*

Cas d'utilisation	Acteurs	Description
Lancer un modèle d'environnement 3D	Utilisateur	- Charger l'environnement dans Gazebo.
Lancer un modèle du robot	Utilisateur	- Charger le robot dans Gazebo.
Gestion de l'environnement	Utilisateur	- Supprimer un robot ou ajouter, déplacer, supprimer un objet.

Tableau 5 : *Description du Diagramme de cas d'utilisation Configuration de l'environnement.*

5.1.3. Diagramme de cas d'utilisation Gestion de l'environnement :

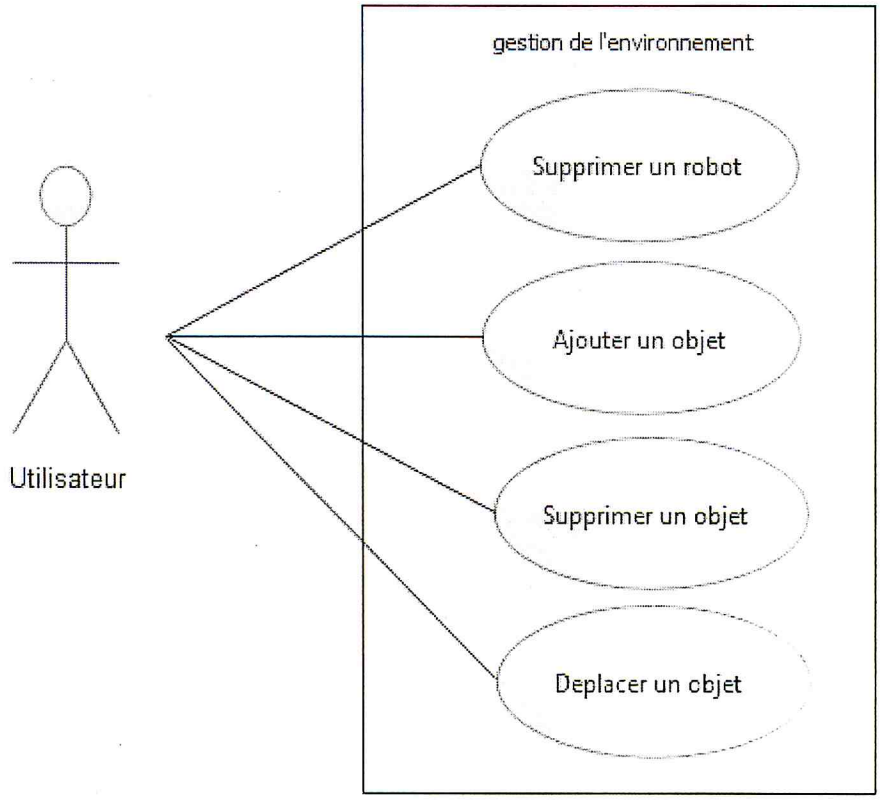


Figure 13 : *Diagramme de cas d'utilisation Gestion de l'environnement.*

Cas d'utilisation	Acteurs	Description
Supprimer un robot	Utilisateur	- L'utilisateur peut supprimer un robot dans l'environnement.
Ajouter un objet	Utilisateur	- L'utilisateur peut ajouter un objet dans l'environnement.
Supprimer un objet	Utilisateur	- L'utilisateur peut supprimer un objet dans l'environnement.
Déplacer un objet	Utilisateur	- L'utilisateur peut déplacer un objet dans l'environnement.

Tableau 6 : *Description du Diagramme de cas d'utilisation Gestion de l'environnement.*

5.1.4. Diagramme de cas d'utilisation Contrôle :

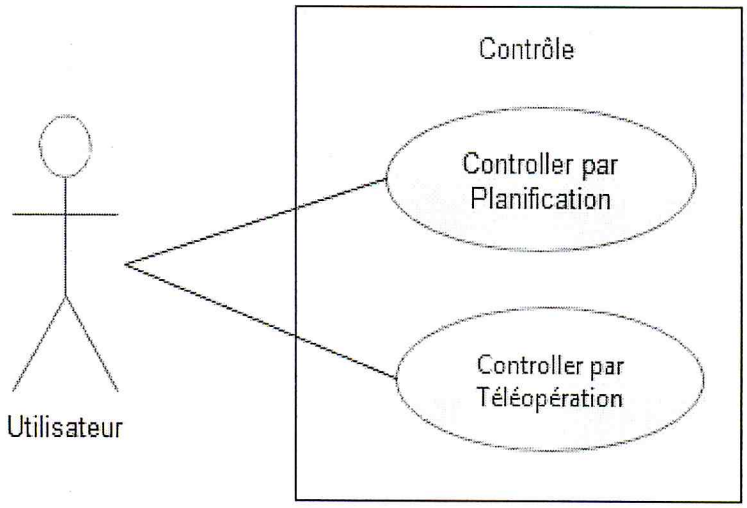


Figure 14 : *Diagramme de cas d'utilisation Contrôle.*

Cas d'utilisation	Acteurs	Description
Controller par Planification	Utilisateur	L'utilisateur exécute un code python pour déplacer le robot à une position cible.
Controller par Téléopération	Utilisateur	L'utilisateur peut contrôler le robot par clavier.

Tableau 7 : *Description du Diagramme de cas d'utilisation Contrôle.*

5.2. Diagramme de classes :

Les diagrammes de classes décrivent les types des objets qui composent un système et les différents types de relations statiques qui existent entre eux [AUD, 06]. Les classes qui composent notre système sont :

- Classe Environnement.
- Classe Objet.
- Classe Robot.
- Classe Capteur.
- Classe camera.
- Classe Capteur.
- Classe Contrôleur.

- Classe planification.
- Classe téléopération.

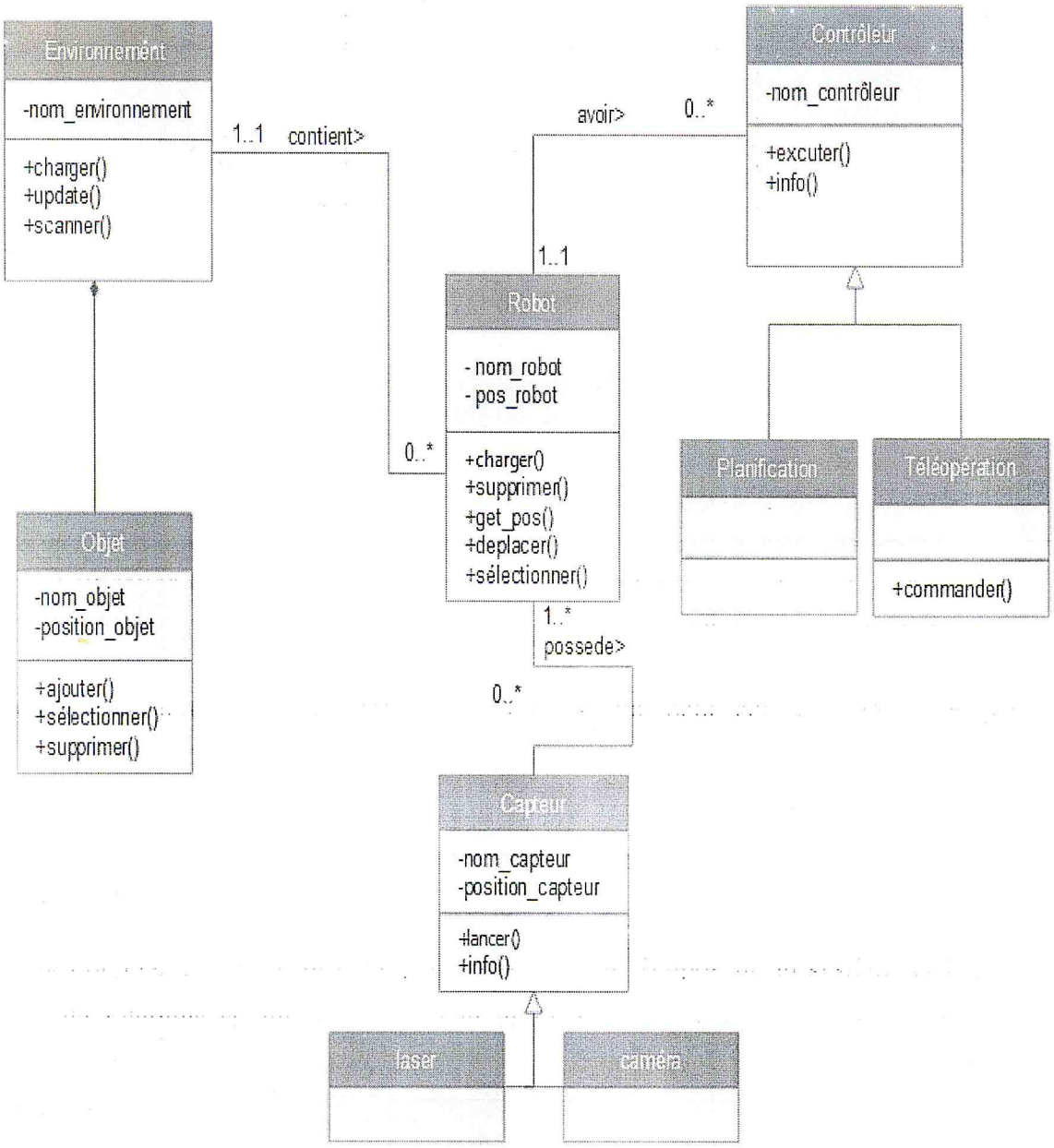


Figure 15 : *Diagramme de classes.*

5.3. Diagramme de séquence

Le diagramme de séquence montre les interactions entre les objets, agencées en séquence dans le temps. Il montre en particulier les objets participants à l'interaction par leurs lignes de vie et les messages qu'ils s'échangent de façon ordonnancée dans le temps [SAL, 11]. Dans ce qui suit, nous présentons les principaux diagrammes de séquence :

5.3.1. Diagramme de séquence Lancement de la simulation :

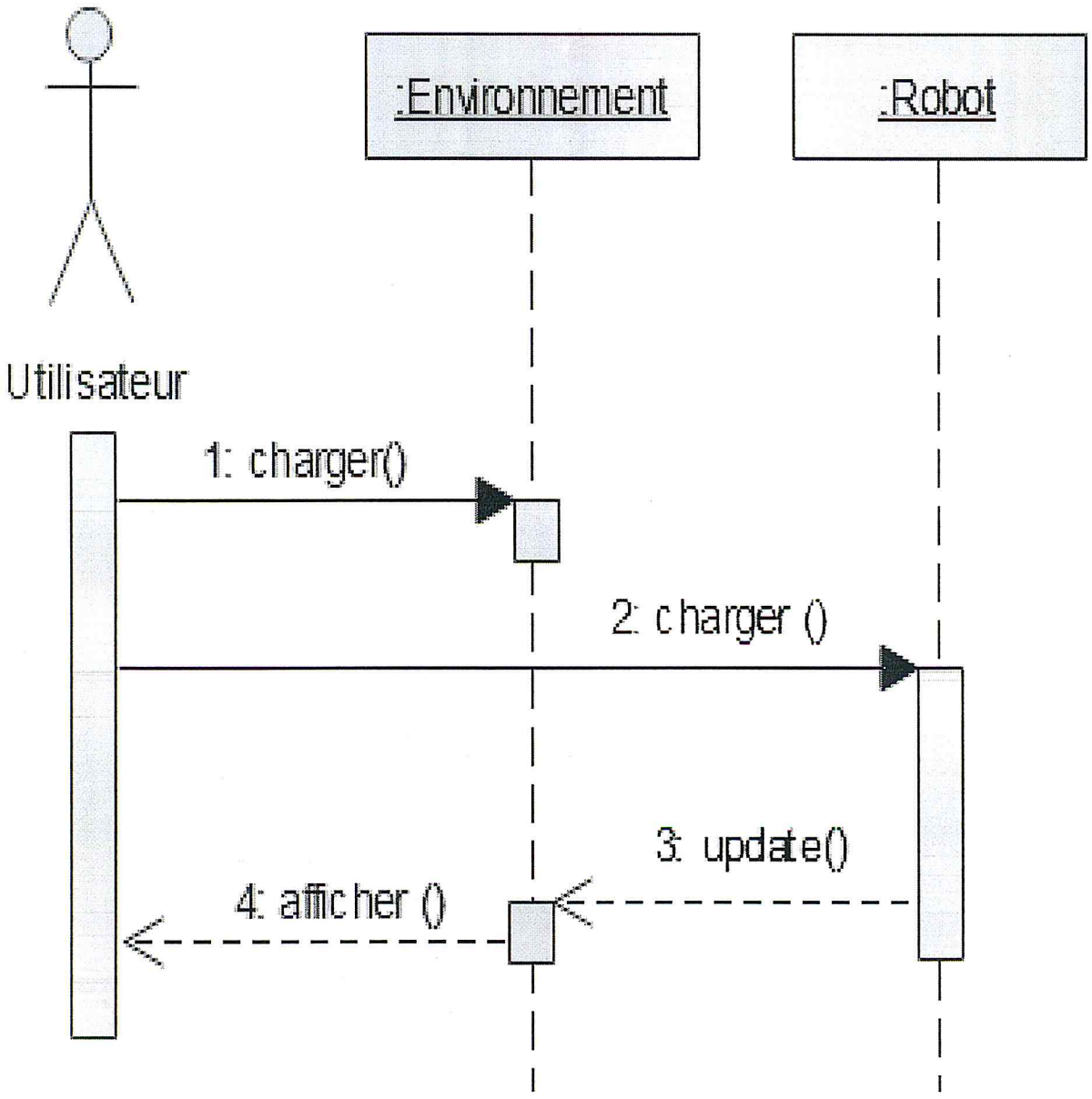


Figure 16 : *Diagramme de séquence Lancement de la simulation.*

5.3.2. Diagramme de séquence Contrôle :

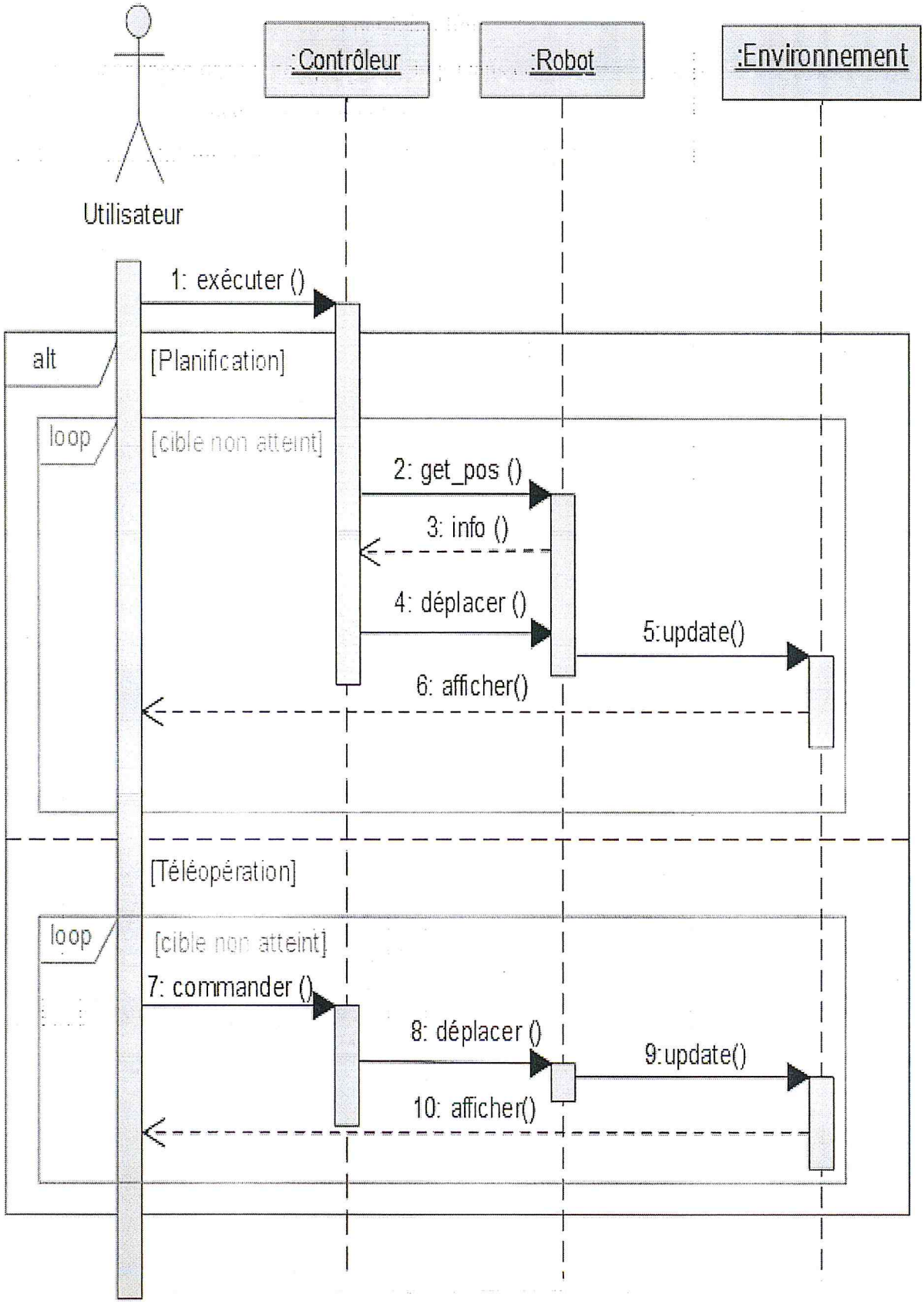


Figure 17 : *Diagramme de séquence Contrôle.*

5.3.3. Diagramme de séquence Fonctionnement de capteur :

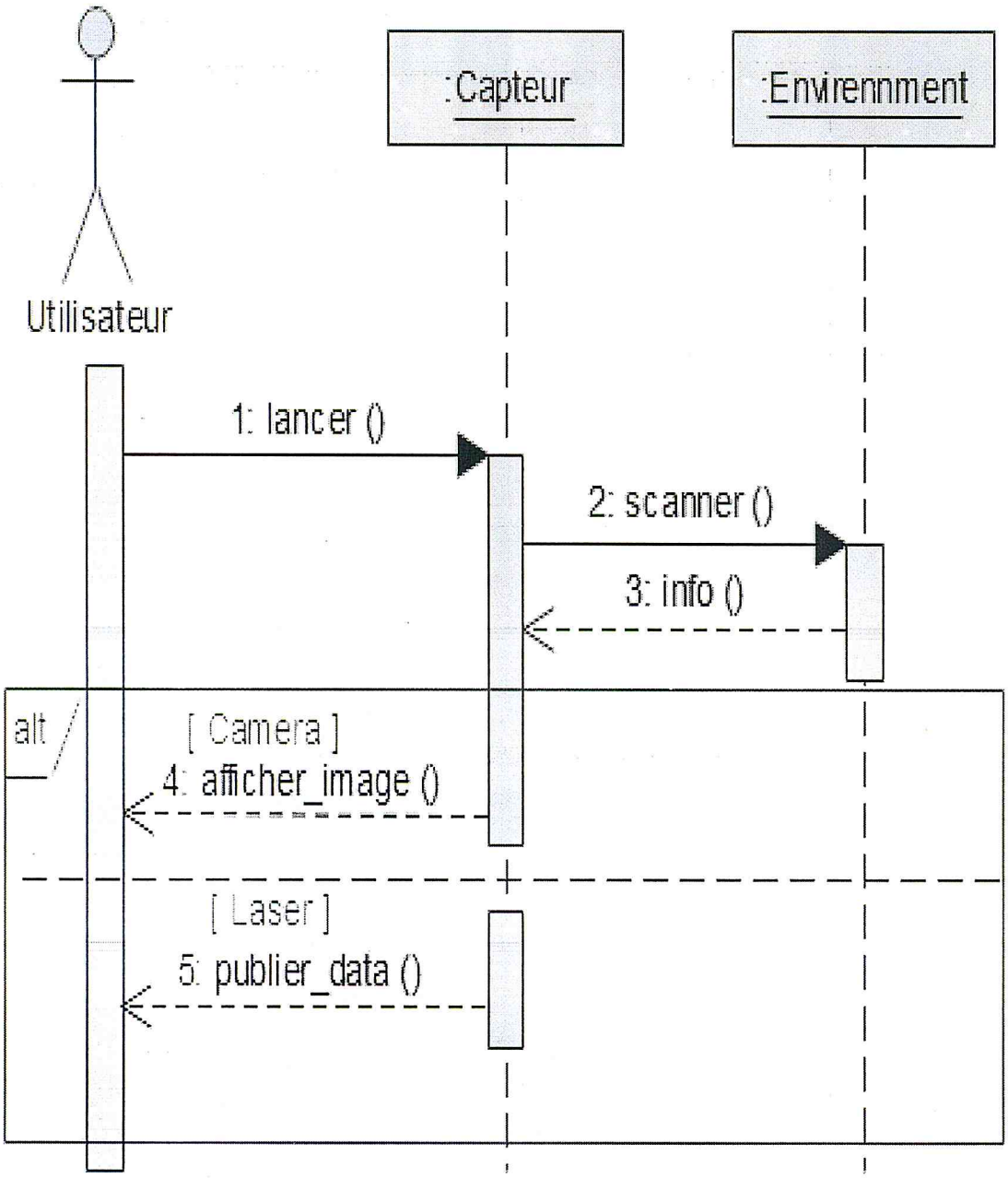


Figure 18 : *Diagramme de séquence Fonctionnement de capteur.*

5.3.4. Diagramme de séquence Gestion de l'environnement :

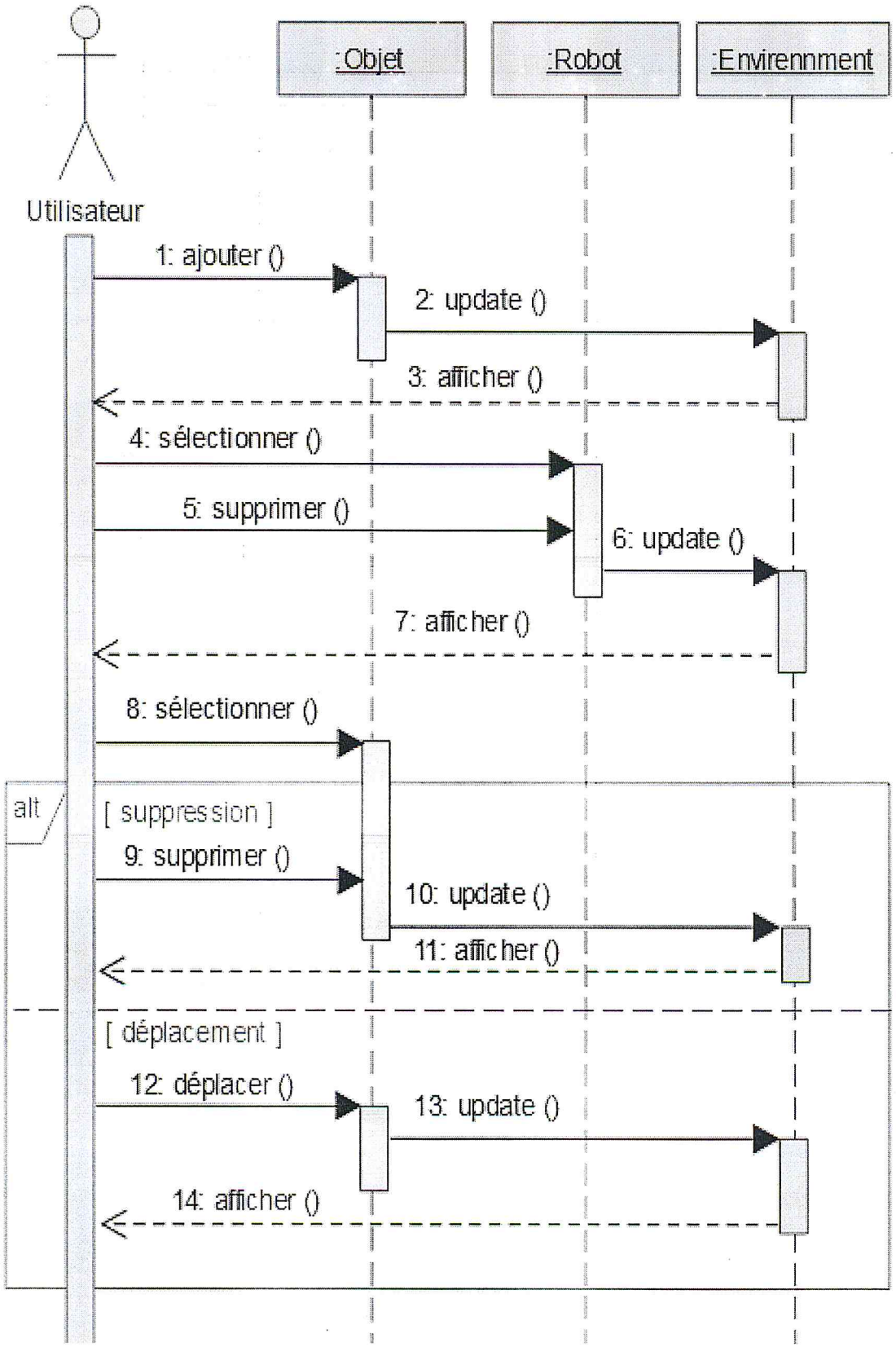


Figure 19 : *Diagramme de séquence Gestion de l'environnement.*

6. Conclusion :

Dans ce chapitre, nous avons défini la conception de notre travail en commençant par la présentation de la problématique, ainsi que les différents objectifs visés à atteindre. Ensuite nous avons présenté la solution que nous avons proposée avec les différents diagrammes du langage de modélisation UML. Dans le chapitre suivant nous allons passer à l'étape de l'implémentation de notre travail.

Chapitre 4 :

Implémentation

1. Introduction:

Après avoir exprimé les objectifs de notre travail et la solution proposée, dans ce chapitre nous allons passer à la présentation des différentes étapes de la Conception géométrique du Robuter/UlM et la création des différents objets qui sont nécessaire pour passer à la simulation.

2. Conception géométrique du Robuter/ULM :

La conception géométrique du robot est divisée en deux étapes : la première étape consiste à dessiner le modèle géométrique du robot réel sur un logiciel de CAO « Solidworks ». Puis en deuxième étape, le modèle conçu sera exporté en utilisant un SolidWorks add-in qui permet d'exporter des parties Solidworks et les assemblages en un fichier URDF, pour pouvoir ensuite mettre le fichier URDF et les maillages exportés dans notre package ROS.

2.1. Dessin du modèle géométrique :

Solidworks permet de modéliser des pièces et leurs assemblages dans un environnement 3D. Voici l'assemblage et les différentes pièces conçu sur Solidworks pour le Robuter/ULM :

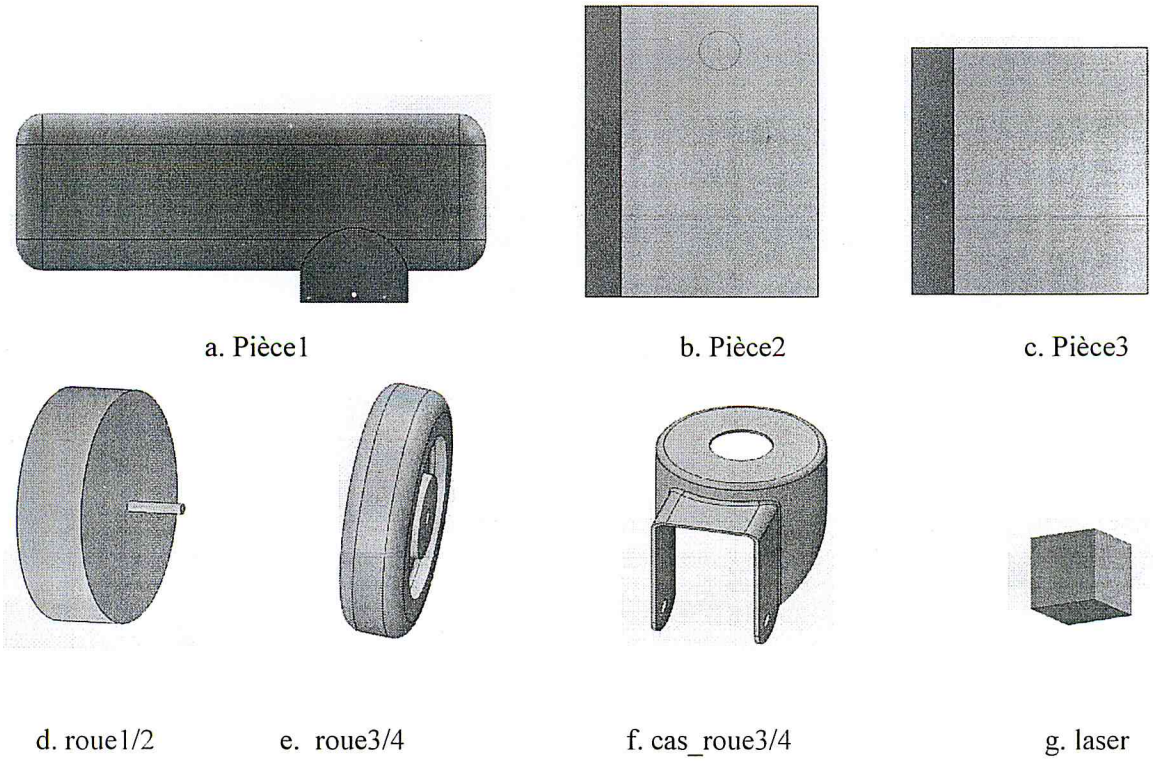
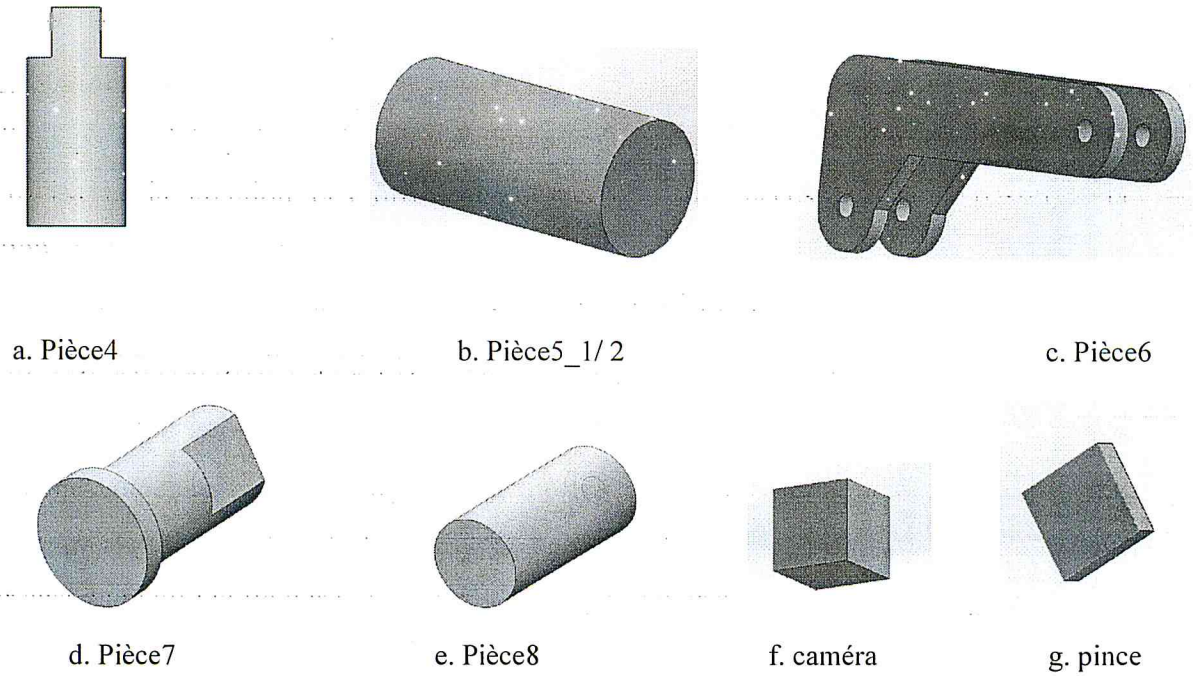


Figure 20 : *Modèles des différentes pièces de la base mobile.*



a. Pièce4 b. Pièce5_1/ 2 c. Pièce6

d. Pièce7 e. Pièce8 f. caméra g. pince

Figure 21 : *Modèles des différentes pièces du bras ultraléger.*

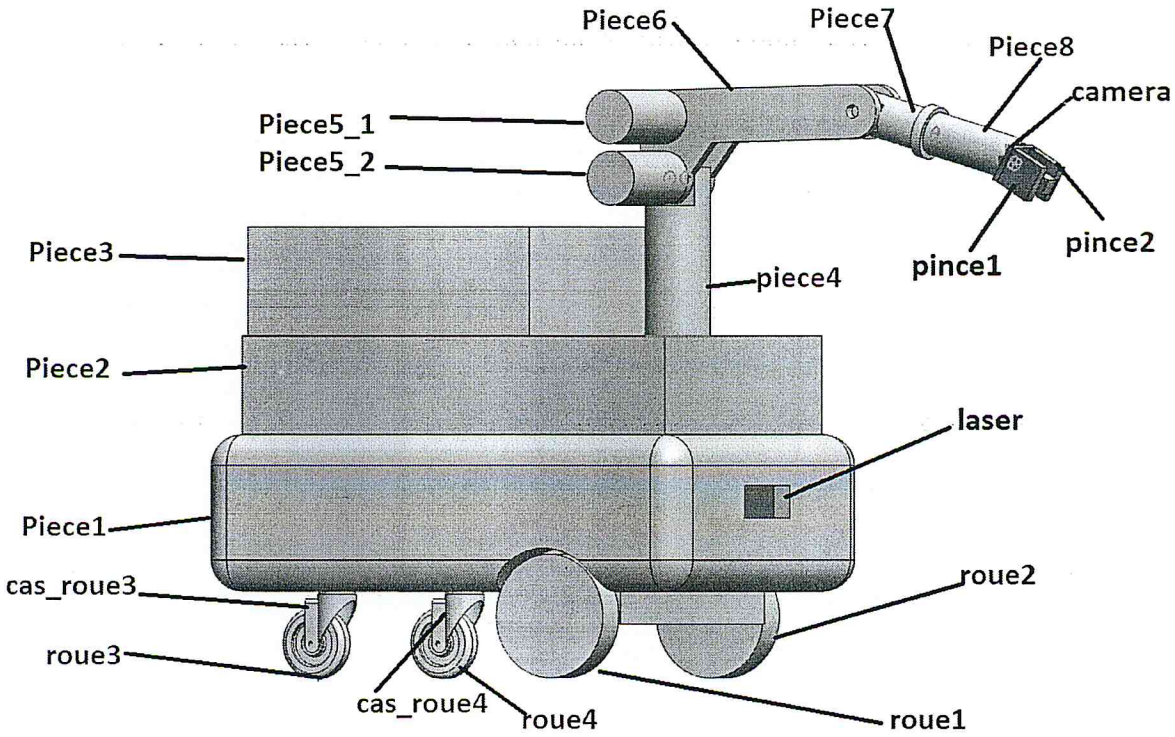


Figure 22 : *Assemblage Robuter/ULM.*

Pour simuler le robot dans le simulateur Gazebo il doit être représenté dans un fichier URDF, d'où la nécessité d'exporter l'assemblage dans Solidworks en format URDF.

2.2. Exportation du modèle géométrique :

L'URDF est composé une structure arborescente des liaisons enfants reliés à des liens parents par une série d'articulations. Afin d'exporter l'assemblage dans un fichier URDF, nous avons défini l'arbre ci-dessous :

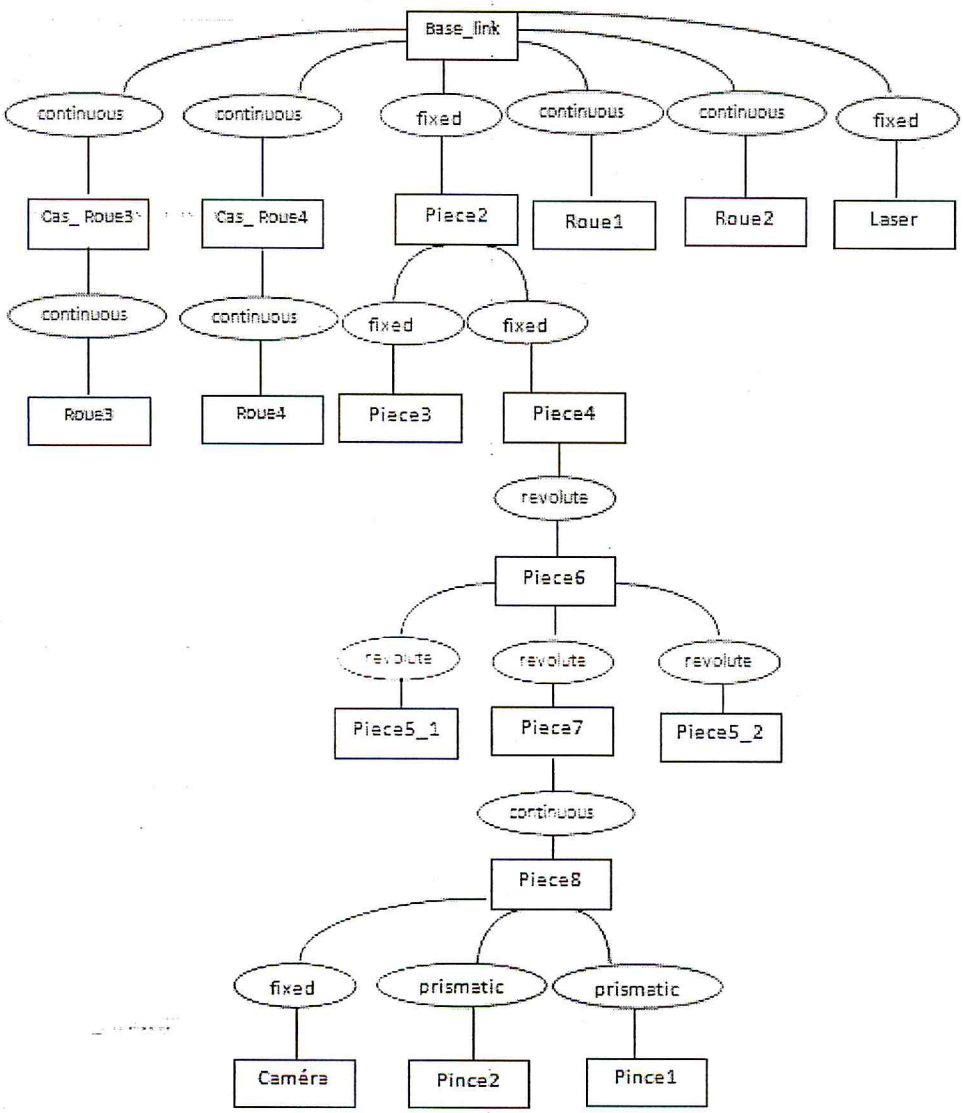


Figure 23 : *Structure arborescente du robot RobuTER/ULM.*

L'exportateur apporte dans la première partie une fenêtre Gestionnaire de propriété pour configurer l'URDF Export, où nous avons configuré chaque lien et construit l'arbre du Figure 23 manuellement par la spécification des liens père-fils entre les différentes pièces du robot.

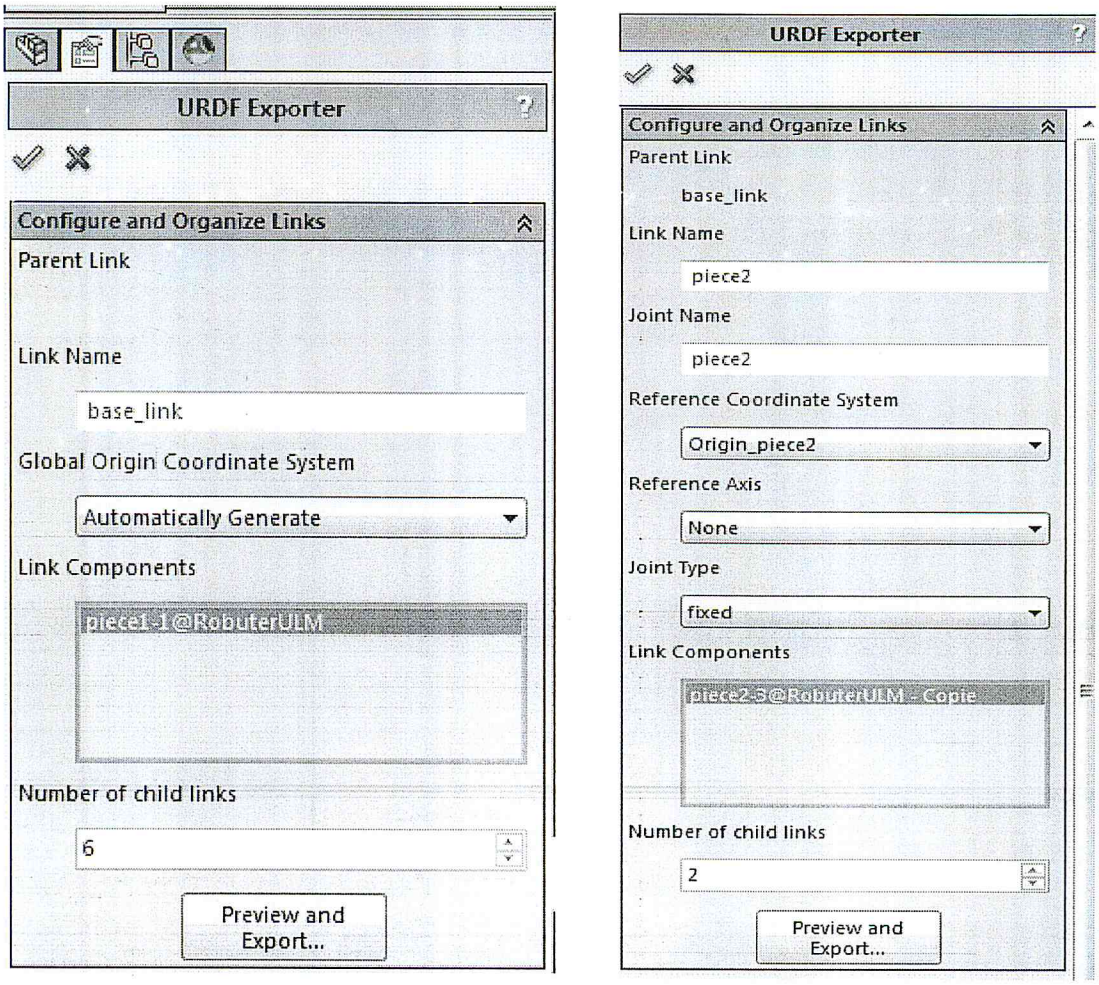


Figure 24 : *les caractéristiques de la partie supérieure de l'exportateur.*

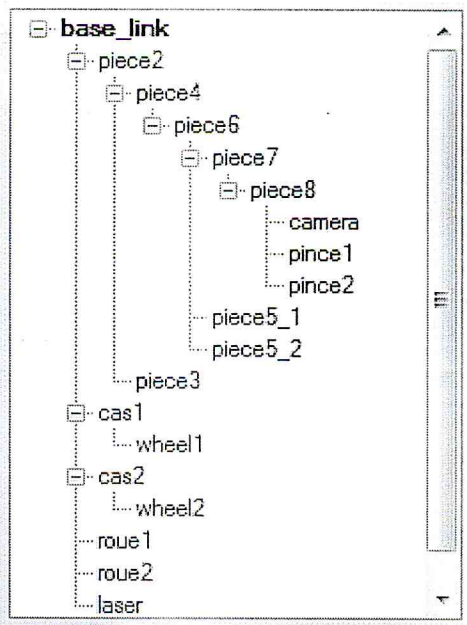


Figure 25 : *La partie inférieure de l'exportateur montre la structure de l'arbre du modèle.*

Dans la deuxième fenêtre de l'exportateur nous avons spécifié les limites des articulations du manipulateur ULM qui sont donnés dans le Tableau 3.

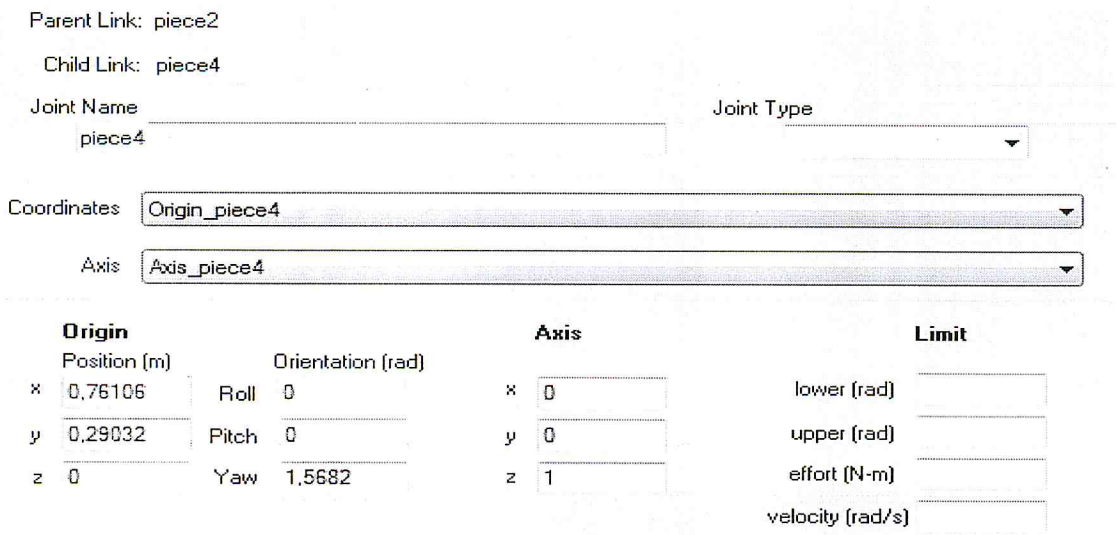


Figure 26 : *La fenêtre des propriétés d'articulation.*

L'exportateur va construire automatiquement les axes de coordonnées, et calculer les propriétés de tous les liens dans l'arbre (les origines des différentes sections, le moment d'inertie, la masse), et met tous ces informations dans le fichier URDF.

3. Création du Meta-package

La communauté ROS a établi des conventions pour les packages qui définissent les robots, leurs consolidations ROS et l'intégration Gazebo. Nous allons essayer de suivre ces conventions pour ce robot. Nous allons créer trois packages :

- **Gazebo :** fournit les fichiers de lancement et de l'environnement pour faciliter le début de la simulation dans Gazebo.
- **Description :** contient le fichier urdf et les mailliages exportés de solideworks.
- **Control :** configure l'interface ROS pour les articulations de notre robot.

4. Création du modèle de simulation

Nous pouvons créer le modèle de simulation pour le RobuTER/ULM en mettant à jour la description du robot existant en ajoutant des paramètres de simulation. Nous allons définir les sections de couleurs, textures, transmission et Gazebo plugin qui sont nécessaires à la simulation.

4.1. Ajout d'éléments Gazebo:

Nous utilisons les éléments Gazebo pour spécifier les propriétés Gazebo comme la couleur, le coefficient de frottement, la gravité pour chaque lien.

```
<gazebo reference="roue1">
   <mu1 value="1.0"/>
   <mu2 value="1.0"/>
   <kp  value="10000000.0" />
   <kd  value="1.0" />
   <fdir1 value="1 0 0"/>
   <material>Gazebo/BlueGlow</material>
   <turnGravityOff>false</turnGravityOff>
 </gazebo>
```

Figure 27 : *Élément Gazebo.*

4.2. Ajout de balises de transmission :

Pour actionner le robot à l'aide des contrôleurs ROS, nous devons ajouter un bloc de transmission pour chaque articulation que nous voulons contrôler pour relier les actionneurs aux articulations. Voici le bloc définie pour la transmission :

```
<transmission name="piece6_trans">
   <type>transmission_interface/SimpleTransmission</type>
   <joint name="piece6">
   <hardwareInterface>EffortJointInterface</hardwareInterface>
   </joint>
   <actuator name="piece6_motor">
     <hardwareInterface>EffortJointInterface</hardwareInterface>
     <mechanicalReduction>1</mechanicalReduction>
   </actuator>
 </transmission>
```

Figure 28 : *Bloc de transmission.*

4.3. Ajout du plugin :

Les plugins Gazebo donnent aux modèles URDF une plus grande fonctionnalité et peuvent attacher des messages ROS et les appels de service pour les données de sortie du capteur et les données d'entrée du moteur, voici les plugins que nous avons ajouté :

a. Gazebo ros control :

Après avoir ajouté les balises de transmission, il faut ajouter le gazebo_ros_control plugin dans le modèle de simulation afin d'analyser les balises de transmission et assigner les interfaces matérielles appropriées et le gestionnaire de contrôle. Le code suivant ajoute le gazebo_ros_control plugin dans le fichier URDF :

```
<gazebo>
  <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
    <robotNamespace>/RobuterULM</robotNamespace>
  </plugin>
</gazebo>
```

Figure 29 : *Gazebo ros control.*

b. Differential drive:

Afin de déplacer le robot dans Gazebo, il faut ajouter un fichier de plug-in Gazebo ROS appelé libgazebo_ros_diff_drive.so pour obtenir le comportement « Differential drive » dans ce robot. Voici l'extrait de code complet de la définition de ce plugin et ses paramètres :

```
  <gazebo>
 <plugin filename="libgazebo_ros_diff_drive.so" name="differential_drive_controller_front">
  <rosDebugLevel>Debug</rosDebugLevel>
  <publishWheelTF>True</publishWheelTF>
  <publishTf>1</publishTf>
  <publishWheelJointState>true</publishWheelJointState>
  <alwaysOn>true</alwaysOn>
  <updateRate>100.0</updateRate>
  <leftJoint>roue2</leftJoint>
  <rightJoint>roue1</rightJoint>
  <wheelSeparation>0.532</wheelSeparation>
  <wheelDiameter>0.227</wheelDiameter>
  <broadcastTF>1</broadcastTF>
  <wheelTorque>30</wheelTorque>
  <wheelAcceleration>1.8</wheelAcceleration>
  <commandTopic>RobuterULM/cmd_vel</commandTopic>
  <odometryFrame>odom</odometryFrame>
  <odometryTopic>odom</odometryTopic>
  <robotBaseFrame>base_footprint</robotBaseFrame>
 </plugin>
</gazebo>
```

Figure 30: *« Differential drive».*

c. Laser:

Nous utilisons le fichier plugin Gazebo ROS appelé libgazebo_ros_laser.so pour simuler le Laser.

```
<gazebo reference="hokuyo_link">
  <sensor type="gpu_ray" name="head_hokuyo_sensor">
    <pose>0 0 0 0 0 0</pose>
    <visualize>false</visualize>
    <update_rate>40</update_rate>
    <ray>
      <scan>
        <horizontal>
          <samples>720</samples>
          <resolution>1</resolution>
          <min_angle>-1.570796</min_angle>
          <max_angle>1.570796</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.10</min>
        <max>30.0</max>
        <resolution>0.01</resolution>
      </range>
      <noise>
        <type>gaussian</type>
        <mean>0.0</mean>
        <stddev>0.01</stddev>
      </noise>
    </ray>
    <plugin name="gazebo_ros_head_hokuyo_controller" filename="libgazebo_ros_gpu_laser.so">
      <topicName>/RoubuterULM/laser/scan</topicName>
      <frameName>hokuyo_link</frameName>
    </plugin>
  </sensor>
</gazebo>
```

Figure 31 : *Laser plugin.*

d. Camera :

Nous utilisons le fichier plugin Gazebo ROS appelé libgazebo_ros_laser.so pour simuler la camera.

```
<gazebo reference="camera_link">
  <sensor type="camera" name="camera1">
    <update_rate>30.0</update_rate>
    <camera name="head">
      <horizontal_fov>1.3962634</horizontal_fov>
      <image>
        <width>800</width>
        <height>800</height>
        <format>R8G8B8</format>
      </image>
      <clip>
        <near>0.02</near>
        <far>300</far>
      </clip>
      <noise>
        <type>gaussian</type>
        <mean>0.0</mean>
        <stddev>0.007</stddev>
      </noise>
    </camera>
    <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
      <alwaysOn>true</alwaysOn>
      <updateRate>0.0</updateRate>
      <cameraName>RoubuterULM/camera1</cameraName>
      <imageTopicName>image_raw</imageTopicName>
      <cameraInfoTopicName>camera_info</cameraInfoTopicName>
      <frameName>camera_link</frameName>
      <hackBaseline>0.07</hackBaseline>
      <distortionK1>0.0</distortionK1>
      <distortionK2>0.0</distortionK2>
      <distortionK3>0.0</distortionK3>
      <distortionT1>0.0</distortionT1>
      <distortionT2>0.0</distortionT2>
    </plugin>
  </sensor>
</gazebo>
```

Figure 32 *: Caméra plugin.*

5. Création de contrôleurs ROS :

Pour déplacer chaque articulation, nous avons besoin d'attacher un contrôleur qui est compatible avec l'interface mentionnée à l'intérieur des balises de transmission dans chaque articulation. Connecter les contrôleurs de robots à chaque articulation consiste à écrire un fichier de configuration pour trois contrôleurs :

- **Joint state controllers :** publie les états des articulations du robot.
- **Joint velocity controllers :** recevoir une vitesse pour chaque articulation.

- **Joint position controllers :** recevoir une position cible pour chaque articulation et peuvent déplacer chaque articulation. La figure ci-dessous représente une partie du fichier de configuration position_control.yaml:

```
RobuterULM:
  # Publish all joint states -----------------------
  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 50

# Position Controllers ---------------------------------
  joint1_position_controller:
    type: effort_controllers/JointPositionController
    joint: roue2
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint2_position_controller:
    type: effort_controllers/JointPositionController
    joint: roue1
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint3_position_controller:
    type: effort_controllers/JointPositionController
    joint: piece4
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint4_position_controller:
    type: effort_controllers/JointPositionController
    joint: piece6
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint5_position_controller:
    type: effort_controllers/JointPositionController
    joint: piece7
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint6_position_controller:
    type: effort_controllers/JointPositionController
    joint: piece8
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint7_position_controller:
    type: effort_controllers/JointPositionController
    joint: pince1
    pid: {p: 100.0, i: 0.01, d: 10.0}
```

Figure 33 : *Joint position controllers.*

6. Création de l'environnement de simulation :

La simulation de l'environnement consiste à reproduire les conditions de l'environnement réel. On les utilise pour évaluer la capacité du robot à supporter les contraintes environnementales. Nous souhaitons de simuler un environnement qui ressemble au laboratoire de la Division Productique et Robotique du Centre de Développement des Technologies Avancées (CDTA). Gazebo offre la possibilité de créer des environnements

semblables à la réalité soit par programme ou par l'intermédiaire de l'interface graphique. Il existe plusieurs modèles et mondes disponibles à Gazebo, que nous pouvons insérer manuellement dans notre environnement tel que des tables, bibliothèques, barrières, et des autres robots…etc. Nous pouvons aussi concevoir notre propre monde dans Gazebo, en utilisant l'éditeur de construction.

Nous avons conçu un modèle qui ressemble au laboratoire dans un fichier « .world » qui contient les différents objets de l'environnement :

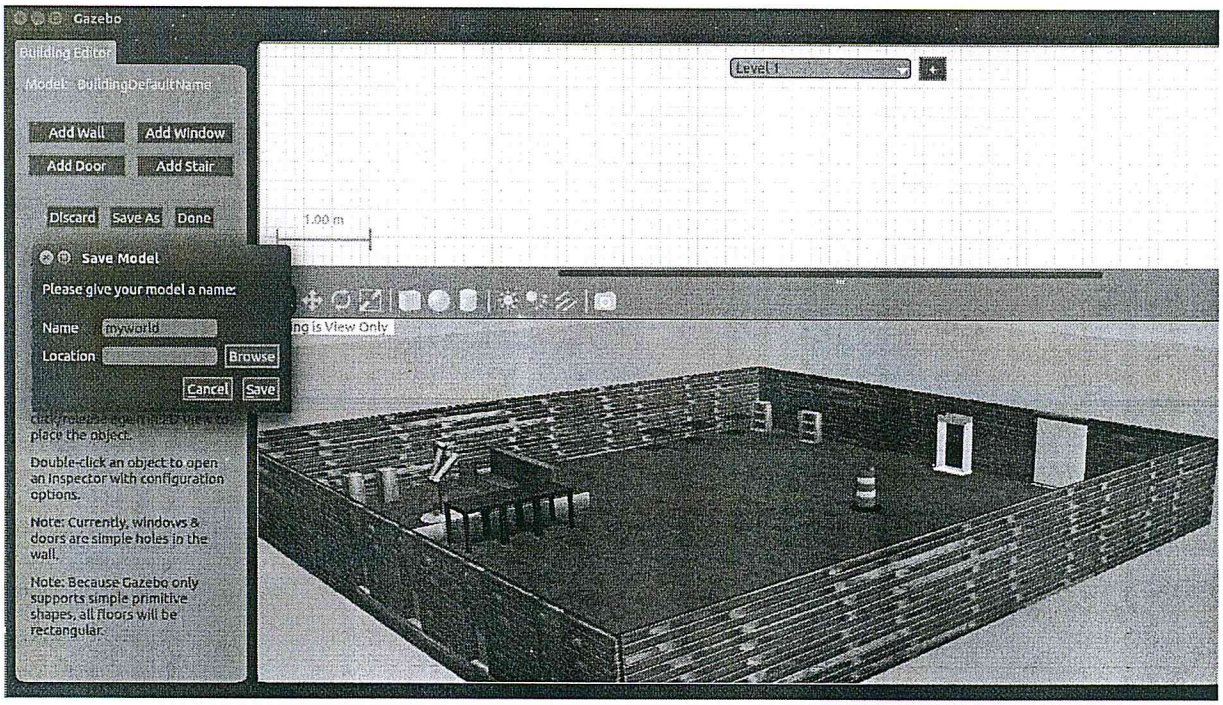


Figure 34 : *l'environnement créer sous Gazebo.*

7. Conclusion :

Dans ce chapitre nous avons construit le fichier URDF pour la simulation dans Gazebo et ajouté les plugins Gazebo ROS nécessaires pour le laser, caméra et le mécanisme du « differential drive » comme nous avons ajouté des contrôleurs et des blocs de transmission à chaque joint. Dans le chapitre suivant, nous allons présenter le robot dans l'environnement de simulation, avec quelques scénarios.

Chapitre 5 :

Test

Et

Validation

1. Introduction

Après les étapes de conception, de proposition des solutions et d'implémentation et d'intégration des différents modules nécessaires à la simulation, nous passons maintenant à l'étape de test et de validation de notre travail. Le présent chapitre décrit le lancement de la simulation et les différents scénarios de validation.

2. Lancement de la simulation :

Pour lancer la simulation, nous lançons un fichier ".launch" qui offre un moyen pratique pour démarrer plusieurs nœuds et "Master", ainsi que d'autres exigences d'initialisation telles que le réglage des paramètres. Il permet de charger le modèle du robot, le modèle de l'environnement, la configuration du contrôleur. Voici le fichier "gazebo.launch" de notre package :

```
<?xml version="1.0" ?>
<launch>
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
  <!-- charger l'environnement créé-->
     <arg name="world_name" value="$(find RobuterULM)/worlds/Environnement.world"/>
    <arg name="debug" value="$(arg debug)" />
    <arg name="gui" value="$(arg gui)" />
    <arg name="use_sim_time" value="true" />
    <arg name="headless" value="false" />
  </include>

   <arg name="robot_name" value="RobuterULM"/>
 <!--charger la description Urdf  sur le Parameter Server-->
 <param name="robot_description"
    textfile="$(find RobuterULM)/robots/RobuterULM.URDF"/>
 <!--charger le modele du robot dans gazebo -->
    <node pkg="gazebo_ros" type="spawn_model" name="spawn_$(arg robot_name)"
         args="-x 0.0 -y 0.0 -z 0.25 -unpause -urdf -param robot_description -model $(arg robot_name)"
         respawn="false" >
    </node>
<!-- charger les configurations du contrôleur d'articulation à partire du fichier YAML sur le Parameter Server-->
  <rosparam file="$(find RobuterULM)/control/config/pos_control.yaml" command="load"/>
<!--charger les configurations du contrôleur dans l'espace de nom RobuterULM-->
  <node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
    output="screen" ns="/RobuterULM" args="joint1_position_controller joint2_position_controller
                                           joint3_position_controller joint4_position_controller
                                           joint5_position_controller joint6_position_controller
                                           joint7_position_controller joint8_position_controller
                                           joint_state_controller"  />

  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher"
       respawn="false" output="screen">
       <param name="publish_frequency" type="double" value="30.0" />
  <remap from="/joint_states" to="/RobuterULM/joint_states" />
  </node>
</launch>
```

Figure 35 : *Fichier de lancement de la simulation.*

Pour contrôler le robot à l'aide de ROS on utilise les topics. Voici une partie de la liste des topics utilisés par le robot simulé:

```
roro@roro-Inspiron-3537: ~
roro@roro-Inspiron-3537:~$ rostopic list
/RobuterULM/camera/camera_info
/RobuterULM/camera/image_raw
/RobuterULM/camera/image_raw/compressed
/RobuterULM/camera/image_raw/compressed/parameter_descriptions
/RobuterULM/camera/image_raw/compressed/parameter_updates
/RobuterULM/camera/image_raw/compressedDepth
/RobuterULM/camera/image_raw/compressedDepth/parameter_descriptions
/RobuterULM/camera/image_raw/compressedDepth/parameter_updates
/RobuterULM/camera/image_raw/theora
/RobuterULM/camera/image_raw/theora/parameter_descriptions
/RobuterULM/camera/image_raw/theora/parameter_updates
/RobuterULM/camera/parameter_descriptions
/RobuterULM/camera/parameter_updates
/RobuterULM/joint1_position_controller/command
/RobuterULM/joint1_position_controller/pid/parameter_descriptions
/RobuterULM/joint1_position_controller/pid/parameter_updates
/RobuterULM/joint1_position_controller/state
/RobuterULM/joint2_position_controller/command
/RobuterULM/joint2_position_controller/pid/parameter_descriptions
/RobuterULM/joint2_position_controller/pid/parameter_updates
/RobuterULM/joint2_position_controller/state
```

Figure 36 : *Partie de la liste des topics.*

Après le lancement de simulation, Nous pouvons voir le modèle du robot dans l'environnement.

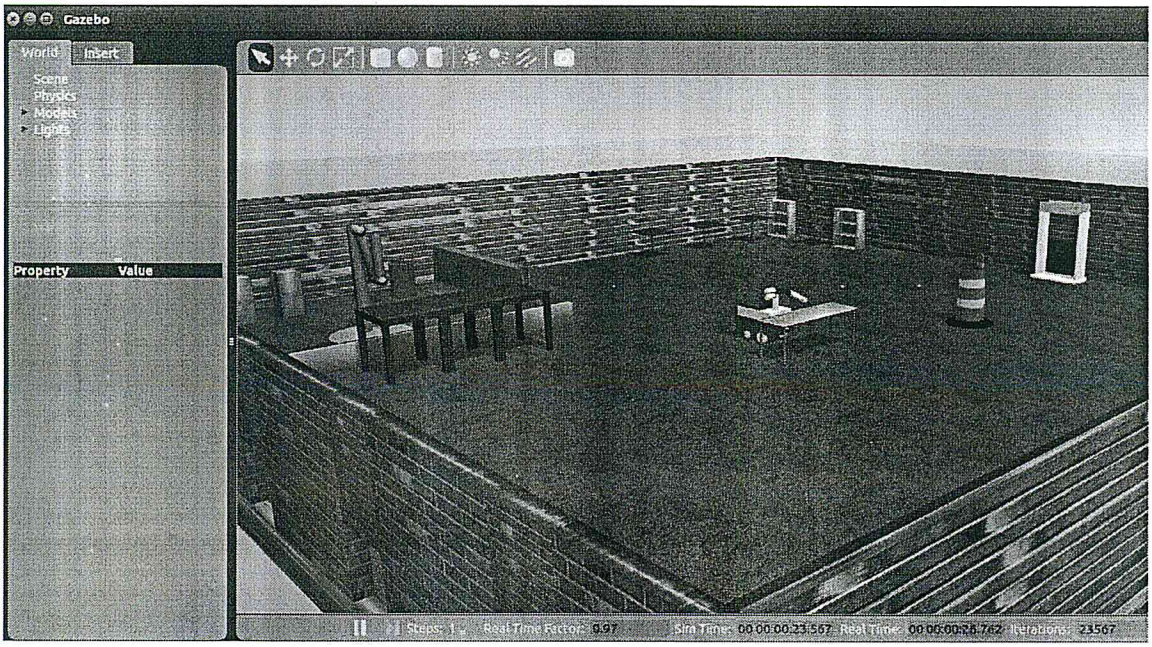


Figure 37 : *RobuTER/ULM dans l'environnement du laboratoire.*

3. Simulation de capteurs:

Nous pouvons voir le fonctionnement du laser en ajoutant des cylindres autour du robot dans l'environnement de simulation.

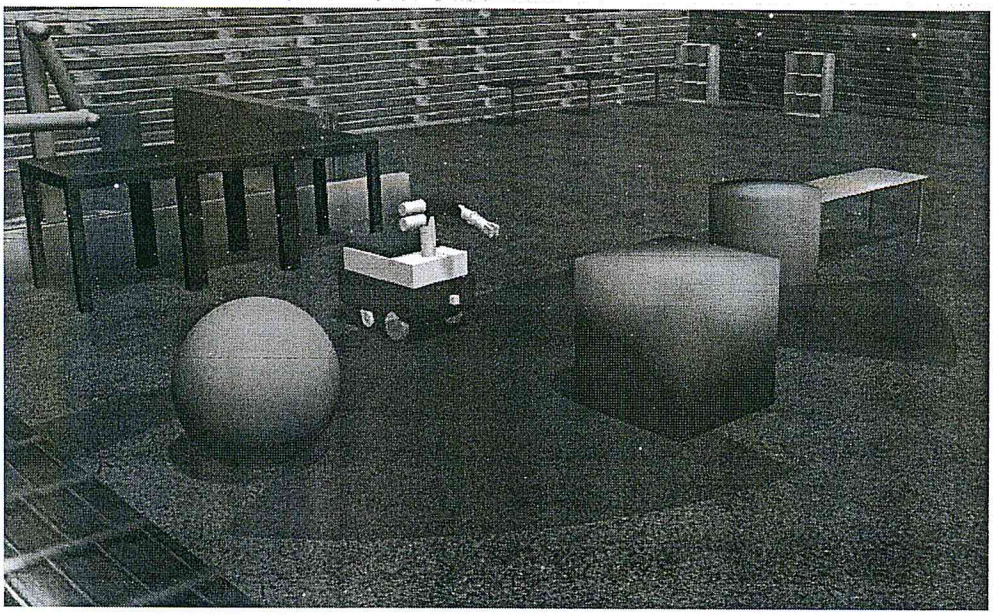


Figure 38 : *Fonctionnement du laser.*

La caméra nous permet de voir l'environnement par les yeux de robots.

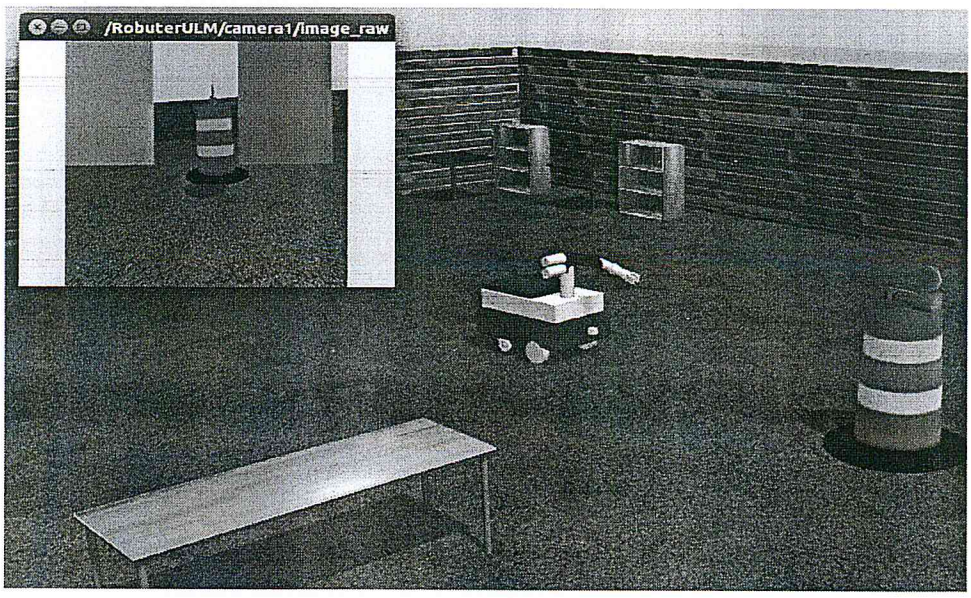


Figure 39 : *Vue de la caméra.*

4. Contrôle du robot mobile

Dans cette partie, nous présenterons les différents contrôles que nous avons utilisés pour contrôler le robot dans Gazebo.

4.1. Contrôle par téléopération

La téléopération permet de contrôler RobuTER/ULM manuellement par clavier. Il existe déjà une implémentation de nœud teleop standard disponible, nous avons réutilisé ce nœud pour contrôler le robot par clavier. La figure ci-dessous montre les instructions de sortie du contrôleur "teleoperation.py" dans le terminal:

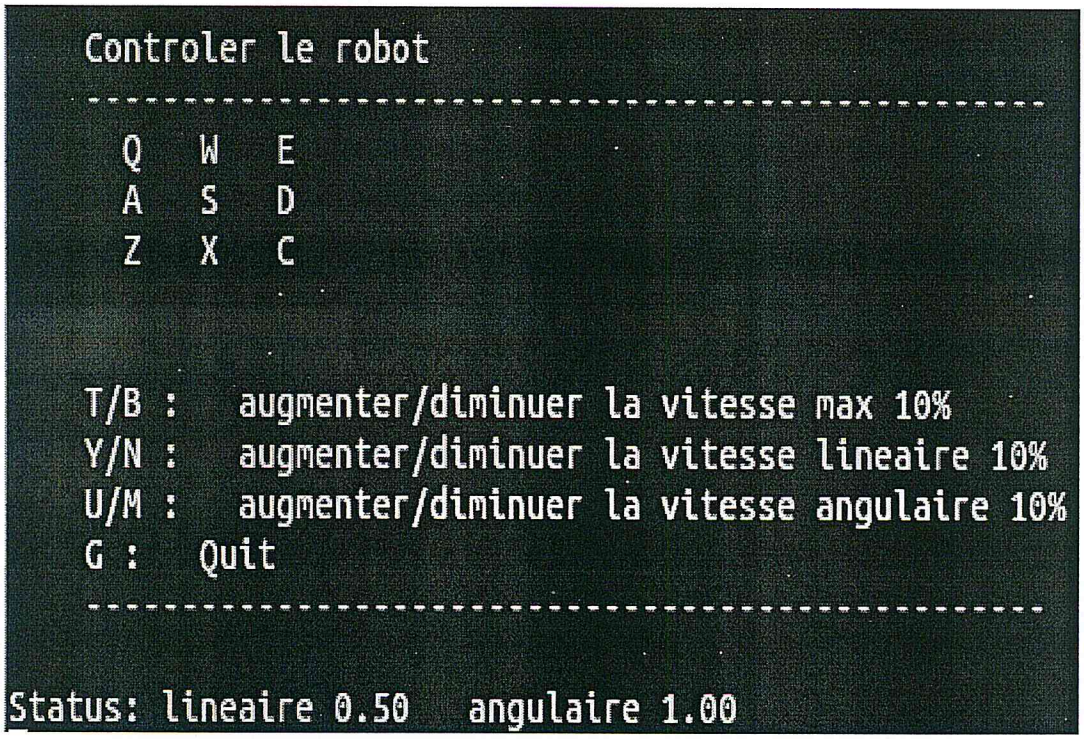


Figure 40 : *Contrôle par téléopération.*

Quand on utilise les touches montré dans la Figure 40 RobuTER/ULM peut tourner, aller vers l'avant et l'arrière et augmenter/diminuer la vitesse.

4.2. Contrôle par planification

Le contrôle par planification consiste à planifier les mouvements du robot, on lui donnant un chemin à suivre. Pour déplacer le robot dans l'environnement d'une position initiale à une position cible, nous exécutons le contrôleur "planification.py".

5. Scénarios de validation

Pour la validation de notre travail, nous avons choisi deux scénarios. Dans le premier scénario, nous allons simuler un seul robot (RobuTER/ULM) et le contrôler. Dans le deuxième scenario nous allons simuler le robot (RobuTER/ULM) avec d'autres robots dans l'environnement. Les différents scénarios traités sont présentés dans ce qui suit :

5.1. Premier Scénario

Dans ce scénario, nous avons lancé la simulation du modèle 3D du RoubuterULM. Nous allons contrôler RobuTER/ULM par :

a. Téléopération:

Lorsque l'on exécute le programme de téléopération, une fenêtre de terminal s'affiche par laquelle nous pouvons contrôler le robot par clavier.

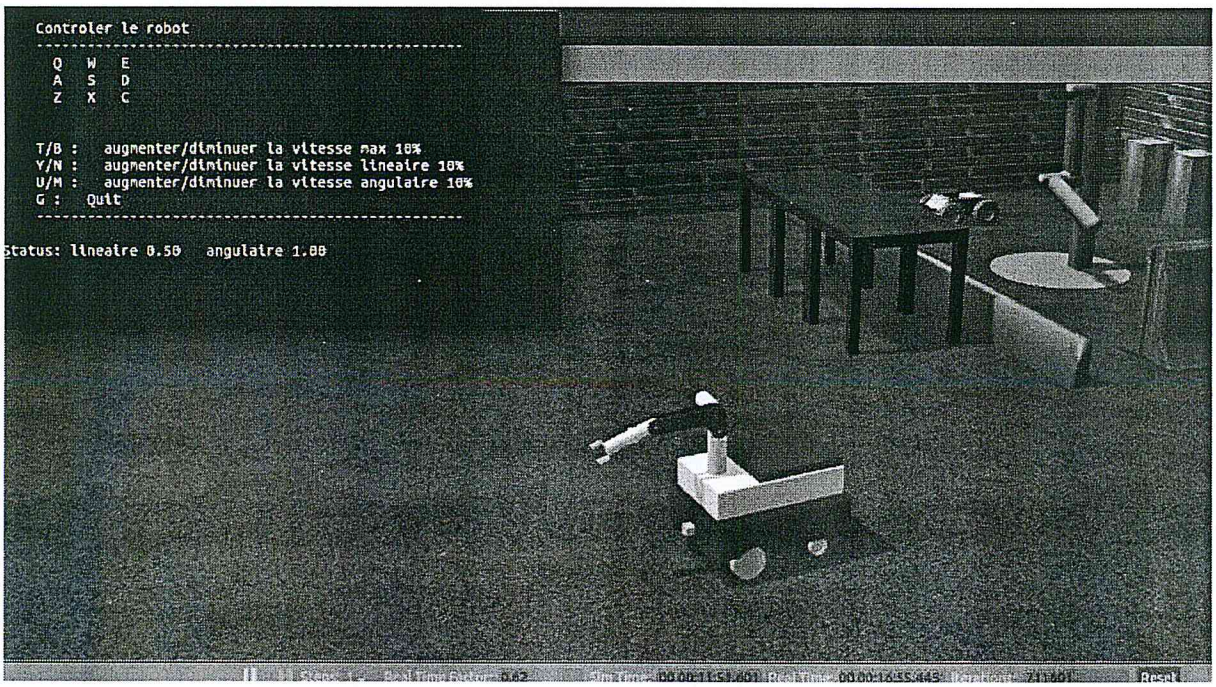


Figure 41 : Déplacement du robot dans Gazebo en utilisant la téléopération.

b. Planification:

Après le démarrage du programme de planification, le robot se déplace dans l'environnement, nous pouvons voir le mouvement du robot dans la figure ci-dessous:

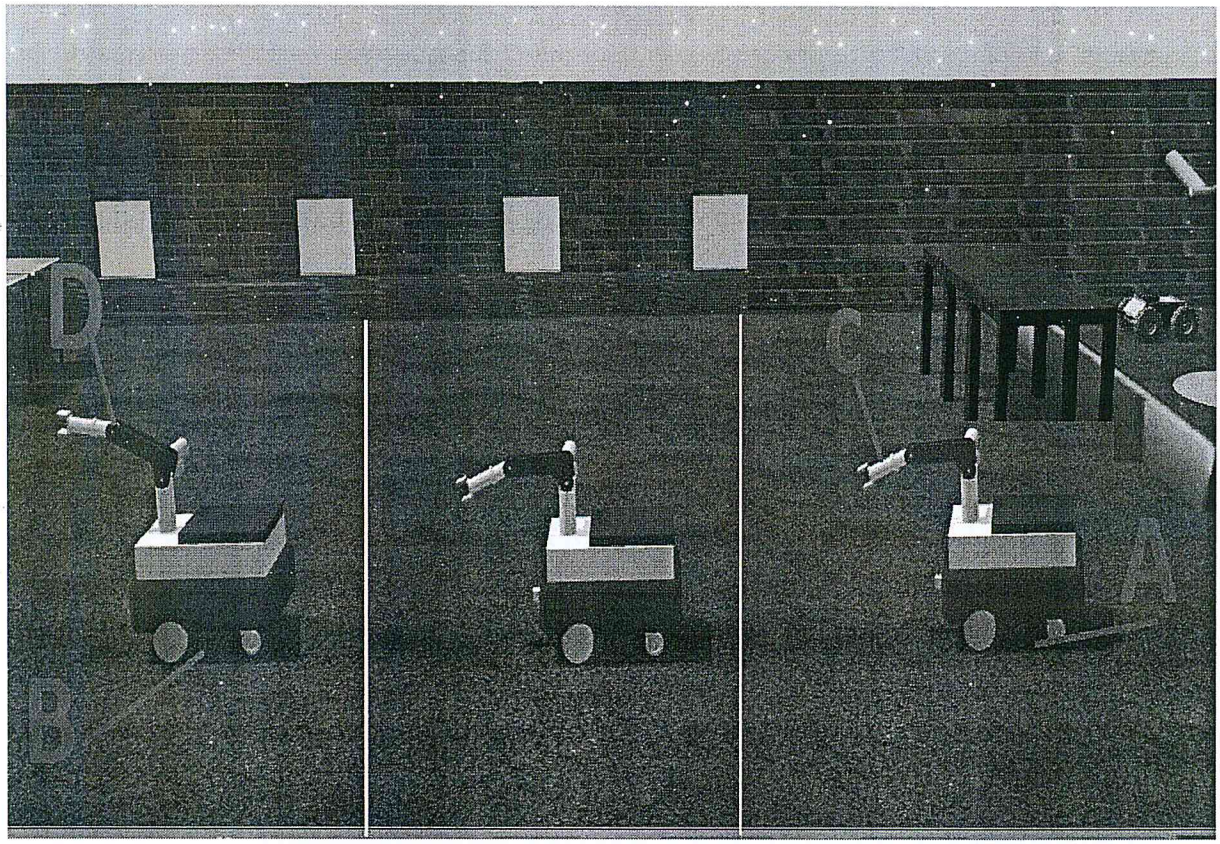


Figure 42 : *Déplacement du robot dans Gazebo en utilisant la planification.*

Pour le déplacement du robot dans l'environnement, nous avons utilisé une fonction « nav », qui publie des commandes de vitesse suivant l'axe des X, et une fonction « get_odom()» pour récupérer la position actuelle du robot mobile le long de son trajet. La base déplace de la position initiale A vers une position finale B et le bras déplace d'une position C à une position D. Une fois le robot arrive au bon endroit (c'est-à-dire il a les mêmes coordonnées que dans le programme, il s'arrête).

5.2. Deuxième scénario

Nous avons fait ce test pour vérifier les performances de la simulation du système multi-robots. Dans ce scénario, nous avons simulé un groupe de robots qui naviguent dans l'environnement (Figure 43), où chaque robot possède son propre contrôleur et topics qui lui permet de déplacer dans l'environnement.

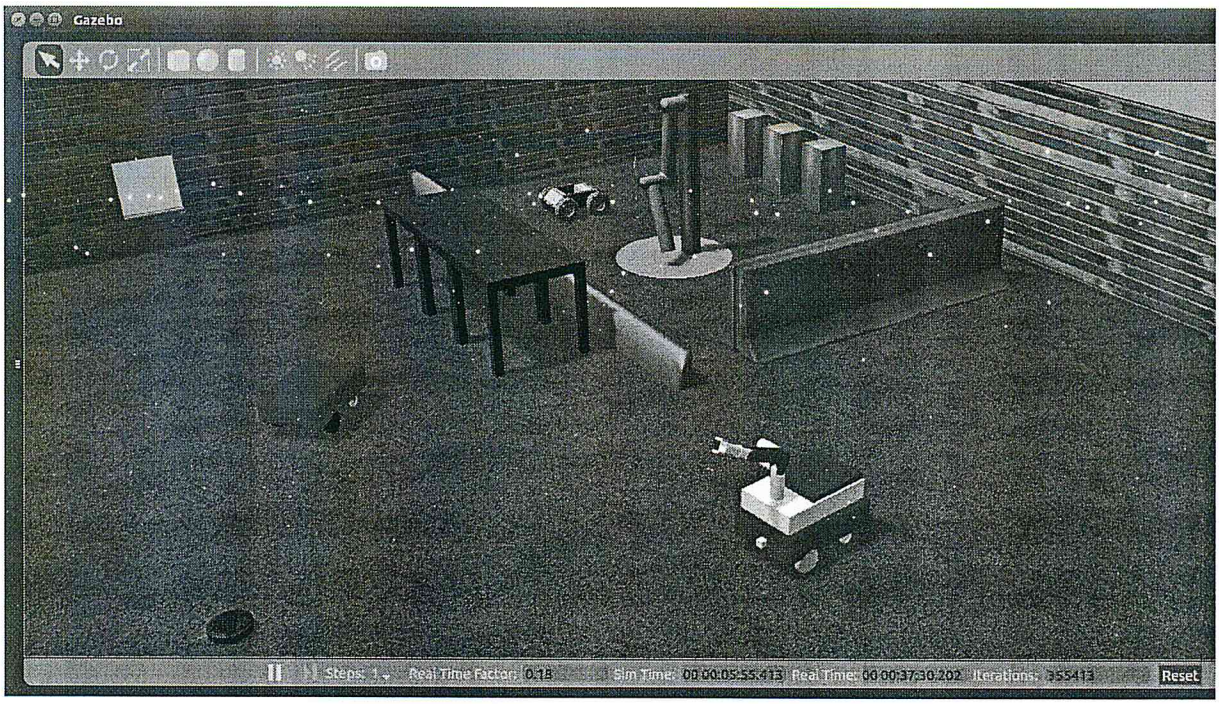


Figure 43 : *Simulation d'un groupe de robots.*

6. Conclusion :

Dans ce chapitre, nous avons présenté les tests de fonctionnement des capteurs du RobuTER/ULM ainsi que les contrôleurs que nous avons utilisé pour contrôler RobuTER/ULM, par la suite nous avons présenté les divers scénarios qui ont été réalisés avec le simulateur Gazebo et ROS (Robot Operating System). En examinant les résultats obtenus, nous avons pu affirmer la validité de la solution que nous avons proposée et implémentée pour la réalisation du notre travail.

Conclusion Générale

Le projet que nous avons présenté dans ce mémoire consiste dans le développement d'un Simulateur 3D pour système Multi-robots hétérogènes.

L'objectif de notre travail était la conception du modèle 3D du manipulateur mobile RobuTER/ULM ainsi que l'environnement du laboratoire du CDTA, puis la simulation de ces derniers dans le simulateur Gazebo à l'aide du ROS (Robot Operating System). Par la suite, pour la création des contrôleurs pour contrôler et déplacer le manipulateur mobile RobuTER/ULM, deux types de contrôle ont été proposés :

- Contrôle par téléopération, l'utilisateur déplace et contrôle le robot par clavier.
- Contrôle par planification, le robot déplace sous une trajectoire programmée d'avance.

Enfin, nous avons proposé différents scénarios d'interaction pour tester et valider notre travail, leurs simulations sur le simulateur Gazebo a présenté de bons résultats.

Durant l'implémentation de notre solution, Nous avons rencontré des problèmes avec l'exportation de l'assemblage en un fichier URDF à cause des valeurs erronées des positions d'articulation, moment d'inertie et des difficultés à apprendre ROS de leurs pages wiki par défaut et les livres qui ne couvrent que les bases d'apprentissage.

Les principaux buts visés ont été atteints, et la majorité des besoins ont été satisfaits. Nous suggérons comme perspectives de notre travail de:

- Appliquer les scénarios élaborés sur le manipulateur mobile expérimental RobuTER/ULM.
- Programmer le robot virtuel pour qu'il puisse éviter les obstacles.
- Etablir la connexion entre le robot réel et virtuel.
- Coordonner les entités du système multi-robots et établir une communication entre elles afin d'élaborer un scénario de type d'interaction entre les robots.

Références Bibliographiques

[ALM, 14] H.H.Alia, P.Mamman, INDUSTRIAL ROBOTICS, In: http://fr.slideshare.net/Gugan7/industrial-robotics-38169689 (2014).

[AUD, 06] L.AUDIBERT, UML 2 De l'apprentissage à la pratique, In : http://laurent-audibert.developpez.com/Cours-UML/?page=introduction-modelisation-objet#L1-4 (2006).

[BAM, 10] bani-Igbida.O and E.Mouaddib, une Histoire de la robotique, In: https://www.u-picardie.fr/~furst/docs/Histoire_robotique.pdf (2010).

[BMF, 12] C. Brambilla, N.Mathews, E.Ferrante, D.Gianni, F.Ducatelle, M.Birattari, M.Luca, M.Gambardella, M.Dorigo, ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems, In: http://link.springer.com/article/10.1007%2Fs11721-012-0072-5 (2012).

[COL, 13] D.Coleman, urdf/XML/joint, In: http://wiki.ros.org/urdf/XML/joint (2013).

[COU, 12] M.Couceiro, MRSim - Multi-Robot Simulator (v1.0), In: http://www.mathworks.com/matlabcentral/fileexchange/38409-mrsim-multi-robot-simulator--v1-0- (2012).

[DAV, 14] C.David, Blender usage in academy fields, In: http://bbug.be/?topic=blender-usage-in-academy-fields#164 (2014).

[DGT, 14] M.Dauchet, J.Ganascia, C.Tessier, A.Grinbaum, L.Devillers, Éthique de la recherche en robotique. Rapport de recherche, n°1. Conception et réalisation graphique, In: http://cerna-ethics-allistene.org/digitalAssets/38/38704_Avis_robotique_livret.pdf (Novembre 2014).

[ERD, 96] P.Erard and P.Déguénon. Simulation par événements discrets, In: https://books.google.dz/books?id=gyYVgdVmRhYC&printsec=frontcover&hl=fr#v=onepage&q&f=false (1996).

[ESP, 00] B.Espinasse, Présentation générale sur le langage de modélisation objet : UML, In: http://www.lsis.org/dea/M6optionD/Exp-GL-UML (2000).

[FIL, 05] D.Filliat, Cours robotique mobile. Cours C10-2, In: http://wwwdfr.ensta.fr/Cours/docs/C102/PolyRobotiqueM,obileENSTA.pdf

(2005).

[FOU, 13] J.Fournier, La robotique médicale, In: http://www.lenouveleconomiste.fr/lesdossiers/la-robotique-medicale-17879/ (2013).

[GON, 13] A.Goncalo, RobotTeamSim – 3D Visualization of Cooperative Mobile Robot Missions in Gazebo Virtual Environment, In: http://ap.isr.uc.pt/archive/RobotTeamSim_dissertacao_Goncalo_Augusto.pdf (2013).

[GUI, 14] M.Guillaumel, Chapitre 0 : Généralités sur la robotique, In: http://www.isir.upmc.fr/UserFiles/File/Morel/CHAPITRE%200%20%20RAPPELS%20ROBOTIQUE.pdf (2014).

[HAD, 11]L. Hadrien, Généralités sur les robots industriels, http://techno.college-pablopicasso.fr/public/3eme/Univers_robotique/Robots_industrie.pdf (2011).

[IME, 13] Bouyoucef.Imene, Coordination de robots pour le transport d'objets, https://espace-ressources.iutsf.org/_media/lissi/rapport_bouyoucef.pdf (2013).

[IVF, 14] S.Ivaldi, P.Vincent et N.Francesco, Tools for dynamics simulation of robots: a survey based on user feedback, In: http://arxiv.org/pdf/1402.7050.pdf (2014).

[KAM, 14] Heddouche.Kamel, Etude et conception d'un Robot marchant, In: http://dspace.univ-biskra.dz:8080/jspui/bitstream/123456789/6616/1/ECDRM.pdf (2014).

[KHU, 11] U.Khurshid, Microsoft Robotics Developer Studio 4 Beta Available For Download, In: https://www.technize.net/microsoft-robotics-developer-studio-4-beta-available-for-download/ (2011).

[LAP, 11] J.Laplace, Présentation de Robot Operating System, In:

http://generationrobots.developpez.com/tutoriels/presentation-robot-operating-system/ (2011).

[LEG, 13] S.LE-GALL, Classification et catégorie de robots, In: http://www.cimi.fr/index.php?view=items&cid=8%3Arobotique&id=73%3Aclassification-et-categorie-de-robots&option=com_quickfaq&Itemid=149 (2013).

[LEN, 15] J.lentin, Mastering ROS for Robotics Programming. In: https://www.geekbooks.me/book/view/mastering-ros-for-robotics-programming (2015).

[MOU, 12] T.Moulard, Optimisation numérique pour la robotique et exécution de trajectoires référencées capteurs, In: https://tel.archivesouvertes.fr/file/index/docid/738999/filename/thesis.pdf (2012).

[NOG, 14] L. Nogueira, Comparative Analysis Between Gazebo and V-REP Robotic Simulators, In: http://www.dca.fee.unicamp.br/~gudwin/courses/IA889/2014/IA889-02.pdf (2014).

[PIL, 15] J.PILLOU,Introduction à UML, In: http://static.ccm2.net/www.commentcamarche.net/contents/pdf/introduction-a-uml-1141-noy5lx.pdf (2015).

[POB, 14] A.Pott, T.Bruckmann, Cable-Driven Parallel Robots. https://books.google.dz/books?id=redKBAAAQBAJ&pg=PA80&lpg=PA80&dq=XDE+simulator+robots&source=bl&ots=jhJKlGclGK&sig=mDXKc_8R5VeT_zHlWWGS3-biRSA&hl=fr&sa=X&ved=0ahUKEwj4su77s5nNAhWLvBQKHQ4yCqwQ6AEILDAC#v=onepage&q=XDE%20simulator%20robots&f=false (2014).

[SAL, 11] A.Saliha, Modélisation objet avec UML : le diagramme de séquence. http://salihayacoub.com/420Ke2/Semaine%209/DiagrammeDesequences.pdf (2011).

[TJL, 13] J.Thiault, H.Ji-Min, J.Lollichon, M.Jade, Les inventions en rapport avec la robotique, In:

https://1dd7761f-a-62cb3a1a-s-sites.googlegroups.com/site/3b4mbony2015/3-petite-histoire-de-la-robotique/sJm4wy6Ka_Vaihg7sYx5xmhxG20.pdf?attachauth=ANoY7cqB5ketmd9mKvByIXZ5gf9DjMTLfdmwE0mS8XHrMT_TaxjfnOIgAkNJTANSVThRAZ-t8c_wXGEhhhjolPJMtRDpPf0i4awdP5CmeWX8XIWhPgFWQZDNgwZ_6w9KZ7Our

CkQKZL8cjoo1dggl2XLoUEMUwooAUmC9uU_0MtRmWwexF8kTxcFl8TOQB8mm-4SoXS9QbkJtdz24EJX82StIZG5QFMqmoapEuKmwWBfxzN9o_L2fvt_OxRjfyUGJNKdX_xTh9pssOHADcWTe8NR_gU0x4ckIg%3D%3D&attredirects=0&d=1 (2013).

[TUA, 10] V.Tuan-Le, Coopération dans les systèmes multi-robots : Contribution au maintien de la collectivité et a l'allocation dynamique de rôles, In: http://sma.lip6.fr/Csma/theses/THESE_LE2010.pdf (2010).

[WEE, 06] I.Weebly, Militaire les robots faut il en avoir peur. Les robots faut il en avoir peur?, In: http://lesrobotsfautilenavoirpeur.weebly.com/la-robotique-militaire.html (2006).

[WIK, 16] Wikipedia contributors, Robot Operating System. In: https://fr.wikipedia.org/wiki/Robot_Operating_System (2011).

[ZIN, 15] L.Zineb, La modélisation d'un système de coopération et communication dans la navigation des robots mobiles, In: http://theses.univ-oran1.dz/document/15201650t.pdf (2015).