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 :

Brahim Bouneb Aymen Sidi Mamar Nassim

En vue d'obtenir le diplôme de master

Domaine : Informatique

Filière : Mathématique et informatique
Spécialité : Informatique
Option : Génie système informatique

Thème

CONTRIBUTION AU DEVLOPPEMENT D'UNE STRATEGIE DE GENERATION DE MOUVEMENT POUR UN ROBOT MANIPULATEUR MOBILE

Organisme d'accueil : Centre de Développement des Technologies Avancées.
Soutenu le : 26/06/2016

Mr. M.Ould Khaoua	Président
Mme. S.Aroussi	Examinatrice
Mr. KAMECH Abdelah Hichem	Promoteur
Mme.AKLI Isma	Encadrante

Promotion
2015 / 2016

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 :

Brahim Bouneb Aymen Sidi Mamar Nassim

En vue d'obtenir le diplôme de master

Domaine : Informatique

Filière : Mathématique et informatique
Spécialité : Informatique
Option : Génie système informatique

Thème

CONTRIBUTION AU DEVLOPPEMENT D'UNE STRATEGIE DE GENERATION DE MOUVEMENT POUR UN ROBOT MANIPULATEUR MOBILE

Organisme d'accueil : Centre de Développement des Technologies Avancées.
Soutenu le : 26/06/2016

Mr. M.Ould Khaoua	Président
Mme. S.Aroussi	Examinatrice
Mr. KAMECH Abdelah Hichem	Promoteur
Mme.AKLI Isma	Encadrante

Promotion
2015 / 2016

Résume

La génération de mouvement est une étape essentielle dans l'exécution des tâches en robotique. Cette étape prend part dans tous le mécanisme du fonctionnement du robot, le système logiciel contrôlant le robot doit respecter le cycle perception, décision et action. Le flot de données s'effectue de manière synchrone relative à l'aptitude du robot, une interaction entre les diffèrent composant logiciel et matériel permet au robot de se localiser dans son environnement, détecter les obstacles et agir en temps réel pour éviter les collisions et arrivé au but. Le but de ce projet est le développement (conception et implémentation) d'une architecture de control de robot pour la génération de mouvement pour un robot manipulateur mobile.

Mot clé : Robot manipulateur mobile, middlewares robotique, Génération de mouvement, Architecture de control.

Abstract

The generation of movements is an essential step in the execution of tasks in robotics. This step takes part in all the robot operation of the mechanism, the software controlling the robot system must respect the cycle perception, decision and action. The flood of data is synchronously on the ability of the robot, this interaction between different software components and hardware allows the robot to locate in its environment, detect obstacles and act in real time to avoid collision reached the goal. The project goal is the development (design and implementation) of a robot control architecture for generating motion for a mobile manipulator robot.

توليد الحركات هو خطوة أساسية في تنفيذ المهام في مجال الروبوتات.

هذه الخطوة تشارك في كل عملية روبوت آلية والنتائج بطريقة مختلفة، برنامج التحكم في نظام الروبوت يجب أن يحترم دورة اتخاذ القرارات والإجراءات اللازمة.

تدفق البيانات يكون بشكل متزامن مع قدرة الروبوت، هذا التفاعل بين مكونات البرامج المختلفة والأجهزة يسمح للروبوت تحديد موقعه في بيئته، كشف العقبات وتعمل في الوقت اللحظي لتجنب الاصطدام والوصول إلى الهدف. هدف المشروع هو تطوير (التصميم والتنفيذ) بنية السيطرة الروبوت لتوليد الحركة للروبوت المناور المتنقل

Remerciements

Avant de vous convier à la présentation de ce travail, l'opportunité nous est donnée de témoigner notre gratitude et notre reconnaissance à toutes les personnes qui par leur aide et leurs encouragements nous ont permis de réaliser ce mémoire.

En premier lieu on remercie monsieur ***kamech Abdellah***
Entant que directeur de mémoire il nous guidés dans notre travail et nous aidés à trouver des solutions pour avancer

Nos vifs remerciements vont à notre encadreur ***Mme AKLI Isma*** *pour son soutien et son encouragement et son précieux temps qu'il nous a accordé*

(CDTA).

Que les membres de ce prestigieux et distingué jury soient assurés de notre gratitude pour nous avoir fait l'honneur d'évaluer notre travail.

Nos remerciements s'adressent à tous les enseignants qui ont contribué à notre formation durant les trois ans d'études à cette université.

Nous tenons à remercier tous nos amis et collègues pour leur soutien moral tout au long de la préparation de ce mémoire.

Spécialement à tous les étudiants de la promotion sortante sans exception.

Dédicaces

A mes très chers parents, sans qui je n'aurais jamais vu la lumière du jour, ni devenu ce que je suis. Puissent-ils trouver en ce modeste Travail toute la gratitude et la reconnaissance d'un fils dévoué.

A mon frères : reda

A ma grand mere, à mes oncles et à mes tantes, à mes cousins et cousines et à toute ma famille

A mes amis: djalil,youcef, mohamed,nabil

A tous mes amis que je n'ai pas pu citer

A mon binôme aymen et toute sa famille.

Je dédie ces moments inoubliables

A tous ceux qui, ne serait-ce qu'une fois, me furent d'une aide aussi petite soit-elle; Qui, de près ou de loin, ont contribué à l'élaboration du présent travail. A tous ceux qui m'aiment et à tous ceux pour qui je compte.

Je dédie le fruit de mon travail

Sidi Mamar Nassim

Dédicaces

A mes très chers parents, sans qui je n'aurais jamais vu la lumière

Du jour, ni devenu ce que je suis. Puissent-ils trouver en ce modeste

Travail toute la gratitude et la reconnaissance d'un fils dévoué.

A ma sœur : HIBA

A mes deux frères : ANES et MOHAMED AMINE

A mes grands-parents, à mes oncles et à mes tantes, à mes cousins et cousines et à toute ma famille

A mes amis: Djalil, Nabil, Mohamed.

A tous mes amis de la promotion 2010

A tous mes amis que je n'ai pas pu citer

A mon binôme Nassim et toute sa famille.

Je dédie ces moments inoubliables

A tous ceux qui, ne serait-ce qu'une fois, me furent d'une aide aussi petite soit-elle; Qui, de près ou de loin, ont contribué à l'élaboration du présent travail. A tous ceux qui m'aiment et à tous ceux pour qui je compte.

Je dédie le fruit de mon travail

AYMEN BRAHIM BOUNEB

TABLE DES MATIERES

Liste des figures

Liste des tableaux

Introduction générale

INTRODUCTION GENERALE

Depuis fort longtemps les êtres humains rêvent de créer des machines intelligentes capables d'effectuer des tâches à leurs places. Or créer une machine pouvant réaliser des tâches que seuls les humains sont normalement capables de les faire n'est pas simple.

En effet, sans toujours y penser, les tâches les plus élémentaires de la vie quotidienne d'humain peuvent devenir extrêmement complexes lorsqu'elles sont analysées de plus près.

Depuis un certain temps déjà, les robots sont entrés dans le monde de l'industrie et de la recherche, mais l'étude et la réalisation des robots sont loin d'être banalisés. Certaines catégories de robots dont les robots mobiles manipulateurs, suscitent encore l'intérêt de recherche.

Les robots manipulateurs mobiles comme tout autre catégorie de robots sont appelés à remplacer l'être humain dans des tâches répétitives difficiles ou à risque. Beaucoup de secteurs profitent actuellement des avancées technologiques dans ce domaine.

Ces robots ont trouvé leurs places aussi bien dans le domaine civil (robotique spéciale, agricole, médicale, travaux public,. . .) que dans le domaine militaire (reconnaissance, surveillance,. . .).

Afin d'être autonome, un robot manipulateur mobile doit posséder de nombreuses capacités, premièrement il doit être capable de percevoir son environnement et de se localiser dans celui-ci. Pour ce faire, ce type de robots possède des capteurs, comme des sonars ou un dispositif à balayage laser servant à mesurer des distances le séparant des obstacles se trouvant à proximité.

Une fois localisé dans son environnement, le robot doit être capable de se déplacer d'un point à l'autre en trouvant des chemins sécurisés afin d'éviter des collisions avec les obstacles.

En plus de pouvoir percevoir globalement, un robot doit souvent être capable d'identifier des objets, de reconnaître des personnes, de lire des indications et même de repérer des symboles graphiques. Ces opérations sont effectuées en analysant des acquises par des caméras installées sur le robot.

Un problème élémentaire de robotique est d'aboutir aux fonctions et aux capacités citées précédemment. A la base de toute action ou évènement survenant dans le robot manipulateur mobile, une architecture est conçue pour gérer les différents composants intervenants dans tout ça. Cette architecture permet de configurer le système logiciel du robot en tenant compte des fonctionnalités que peut offrir le robot et les contradictions pouvant surgir.

Chapitre I

Généralité sur la robotique

CHAPITRE I : GENERALITE SUR LA ROBOTIQUE

I.1 Introduction

Dans ce premier chapitre, nous présentons une vue générale sur les robots mobiles ,les robots manipulateurs et les robots manipulateurs mobiles (définition, domaines d'application, et les composants d'un robot manipulateur mobile).

I.2 Robotique général

Le terme de robotique est apparu en 1942 dans le cycle universellement connu rédigé par *Isaac Asimov* et intitulé *"Les robots"[1].*

Une définition concise de ce que peut être un robot est : " Machine programmable qui imite des actions d'une créature intelligente."[1]

Un robot est un système mécanique poly-articulé mû par des actionneurs et commandé par un calculateur qui est destiné à effectuer une grande variété de tâches [1].

I.2.1 Historique :

Au cours de l'histoire on peut distinguer 3 types de robots correspondant en quelques sortes à l'évolution de cette "espèce" créée par l'homme [1].

Le premier type de machine que l'on peut appeler robot correspond aux "Automates". Ceux-ci sont généralement programmés à l'avance et permettent d'effectuer des actions répétitives.

Le second type de robot correspond à ceux qui sont équipés de capteurs (en fait les sens du robot). On trouve des capteurs de température, photo électronique, à ultrasons pour par exemple éviter les obstacles et/ou suivre une trajectoire.

Ces capteurs vont permettre au robot une relative adaptation à son environnement afin de prendre en compte des paramètres aléatoires qui n'aurait pu être envisagés lors de leur programmation initiale.

Ces robots sont donc bien plus autonomes que les automates mais nécessitent un investissement en temps de conception et en argent plus conséquent.

Enfin le dernier type de robot existant correspond à ceux disposant d'une intelligence dite "artificielle" et reposant sur des modèles mathématiques complexes tels que les réseaux de neurones.

En plus de capteurs physiques comme leurs prédécesseurs, ces robots peuvent prendre des décisions beaucoup plus complexes et s'appuient également sur un apprentissage de leurs erreurs comme peut le faire l'être humain. Bien sûr il faudra attendre encore longtemps avant que le plus "intelligent" des robots ne soit égal, tant par sa faculté d'adaptation que par sa prise de décisions, à l'homme.

En 1968, l'institut construit le premier robot mobile capable de voir shankey est le premier robot capable de penser et de réagir au changement dans son environnement.

C'est une machine à roues connectée à un gros ordinateur et qui évolue dans un environnement de cubes et de pyramides de tailles et de couleurs différentes. Ses moyens de perception sont essentiellement une caméra qu'il lui permet d'acquérir des images de son environnement. Quant à ses performances, ça lui demande une cinquantaine de minutes pour effectuer sa mission [3].

I.3 Composant des robots

Un robot est composé de quatre parties principales.

En voici la description :

- Une structure mécanique qui sera le squelette du robot. Une attention particulière doit être portée aux articulations car celles-ci doivent permettent un débattement assez important relatif à l'utilisation voulue.
- Le second élément correspond aux servo-moteurs qui vont permettre au robot d'effectuer réellement ses actions. Ces servo-moteurs seront commandés par la partie commande en interaction avec les informations transmises par les capteurs. Le terme "servo" induit en effet un asservissement effectué en fonction d'une comparaison avec le résultat souhaité et la réalité extérieure.
- La troisième partie composante d'un robot correspond aux différents capteurs sensoriels équipant le robot pour une application particulière.
- Enfin le cerveau : La partie commande. C'est cette partie qui va permettre au robot d'analyser les données provenant des capteurs et d'envoyer les ordres relatifs aux servomoteurs. La partie commande est matérialisée physiquement par le microcontrôleur.

décision markoviennes [8], à l'aide de techniques d'échenillage de Monte Carlo [9] ou avec d'autres méthodes.

I.3.2.ii Vision : en analysant les images captées par les caméras, on peut extraire une multitude d'informations. Par exemple, à l'aide d'un algorithme de segmentation [10], on peut reconnaître un objet de couleur en plus d'estimer sa position relative par rapport à la vue de la camera. À l'aide de techniques de vision traditionnelles [11], il est aussi possible d'estimer certaines distances dans l'environnement. On peut aussi reconnaitre des symboles, des caractères et lire des messages [12].

I.3.2.iii Navigation : un module de navigation est responsable de déplacer un robot de sa position courante vers une destination désirée de façons efficace en toute sécurité, libre de toute collision avec l'environnement. En plus d'inclure des fonctions de perception de l'environnement et de localisation.

I.3.2.iv Module de fusion d'informations capteurs : le module de gestion des
Capteurs est un module qui permet de récolté de toutes les informations concernant les capteurs existants installés sur la base mobile et le bras manipulateur (Capteurs position, capteur d'effort, etc. . .)[13].

I.4 Robot mobile

La robotique mobile est la branche d robotique concernée par les systèmes de robots mobiles qui sont capables de naviguer dans un environnement ou dans un terrain.

Robotique mobile et les robots sont utilisés principalement dans la recherche sur la navigation et l'exploration, avec des applications pour véhicules guidés autonomes.

Des recherches récentes dans les systèmes basés sur le comportement a utilisé des
Robots à pattes comme analogues mécaniques des insectes et des animaux simples [17].

Définition : C'est une Plate-forme mobile avec une grande mobilité au sein de son environnement (air, terre, sous l'eau) et un n système avec les caractéristiques fonctionnelles suivantes :

- Mobilité : Mobilité totale par rapport à l'environnement
- Un certain niveau d'autonomie : Interaction humaine limitée
- la capacité de perception : Détecter et faire réagir dans le milieu

Un robot mobile est une combinaison de divers facteurs physiques (hardware) et calcul (logiciels) composants [17].

En terme de Matériel composants d'un robot mobile peuvent être pris en considération comme un ensemble de sous-systèmes pour :

-locomotion : comment le robot se déplace à travers son environnement.

-raisonnement : comment le robot mappe les mesures en actions.

-communication : comment le robot communique avec l'extérieur opérateur.

-détection : détecter comment le robot mesure les propriétés de lui-même et son Environnement.

En terme de logiciel composants, un ensemble de sous-systèmes sont responsable de: - Planification

-Planification dans ses divers aspects.

I.5 Robot manipulateur

Un robot manipulateur est un mécanisme à commande électronique, constitué de plusieurs segments, qui effectue des tâches en interagissant avec son environnement. Ils sont aussi communément appelés bras comme robotiques [7].

Manipulateurs de robots sont largement utilisés dans le secteur de la fabrication industrielle et également dans beaucoup d'autres applications spécialisées (par exemple, le Canadarm a été utilisé sur navettes spatiales pour manipuler des charges utiles).

L'étude des robots manipulateurs implique de traiter avec les positions et orientations des différents segments qui composent les manipulateurs. Ce module présente les concepts de base qui sont nécessaires pour décrire ces positions et orientations de corps rigides dans l'espace et effectuer les transformations de coordonnées.

Les manipulateurs sont constitués d'un ensemble de biellettes et d'articulations. Les liens sont définis comme étant les sections rigides qui composent le mécanisme et les articulations sont définis comme étant la liaison entre deux maillons.

Le périphérique connecté au manipulateur qui interagit avec son environnement pour effectuer des tâches est appelé l'effecteur [7].

I.6 Robot manipulateur mobile

Un manipulateur mobile consiste en une base mobile sur laquelle sont montés un ou plusieurs manipulateurs. Le robot peut accomplir les tâches les plus courantes de la robotique qui nécessitent locomotion et de manipulation des capacités [34].

qui nécessitent locomotion et de manipulation des capacités [34].

Ces robots autonomes doivent effectuer des tâches prévue dans des environnements complexes, inconnus et changeants avec détection, perceptive, l'acquisition de connaissances, l'apprentissage, inférence, la prise de décision et la capacité d'agir en utilisant uniquement leur calcul physique limitée et ressources avec une intervention humaine réduite [7].

Un robot manipulateur mobile est une association physique de deux robots, un robot manipulateur monté sur un robot mobile. Il contient la navigation d'une base mobile et le contrôle d'un robot manipulateur statique, la caractéristique principale de ces robots réside dans la flexibilité de leur espace de travail contrairement à l'espace de travail limité d'un manipulateur fixe.

Cette caractéristique conféré au robot la possibilité de couvrir un plus important Champ d'action, ces robots ont des applications dans de nombreux domaines comme la manipulation et le transport de pièces d'un endroit à un l'autre, l'exploitation minière, la foresterie, la construction...etc. L'environnement manufacturier à l'environnement de l'être humain (bureau, foyer, hôpitaux) car ils sont particulièrement bien adaptés à certains tâches réalisées par l'être humain. L'une des approches générale est de considérer la mobilité comme un supplément au robot manipulateur.

I.7 Domaines d'application

Véhicules routiers : pour lesquels sont développés des systèmes d'aides à la conduite, de pilotage automatique et/ou de télé conduite [14].

Robotique agricole : avec l'automatisation des engins de récolte qui nécessite des systèmes de guidage sur structure végétale et les véhicules dédié à l'acquisition automatique des donnés en milieu naturel. Il aura fallu trois longues années à Christophe Millot et Guy Julien pour développer le robot V.I.N **(voir Figure I.1)**.

"Le plus grand défi aura été d'arriver à faire comprendre aux caméras ce qu'elles voient et comment correctement l'interpréter" commente Guy Julien [16].

Figure 1.1-- le robot LadyBird

Robotique de service : pour le transport, le nettoyage ou la surveillance des locaux, en milieu agroalimentaire, industriel, hospitalier, etc.

les robots de service font également déjà partie intégrante de notre vie courante : métro automatique, moniteur médical, caisse enregistreuse intelligente,... Autant de systèmes qui assistent et facilitent notre quotidien [14].

Après l'armée et l'industrie, une nouvelle génération de robots est en train de faire son apparition dans les foyers et les entreprises : les robots de services, professionnels ou personnels. Aujourd'hui, ils servent essentiellement pour le ménage ou l'entretien, des jardins, des robots compagnons pour l'éducation ou le loisir, sont aussi proposées au marché grand-publique. Demains ils vont connaître un essor spectaculaire.

Applications médicales : Les robots semblent avoir de l'avenir à l'hôpital. Robodoc (**voir Figure I.2**) est largement vendu aux USA et aide à réaliser certaine opération de chirurgie. Le robot infirmier est encore en projet.

Le cyber-squelette HAL aide les personnes à se deplacer.et le robot patient permet aux futures chirurgiens-dentistes d'apprendre à soigner sans faire de dégâts.

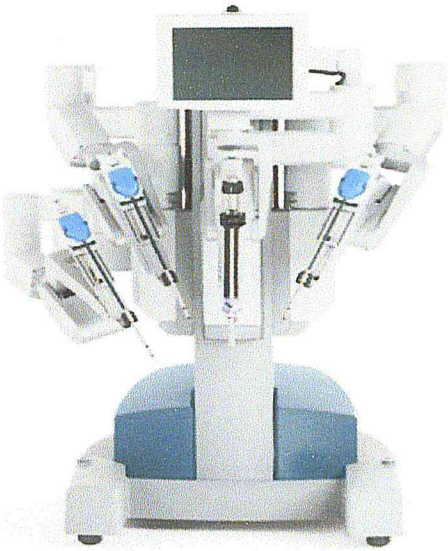


Figure 1.2—Robodoc[14]

Robotique sous-marine : Pour l'automatisation des opérations de maintenance Sur les structures offshores, les taches de reconnaissance, d'exploitation et d'acquisition de mesures sur environnement [14].

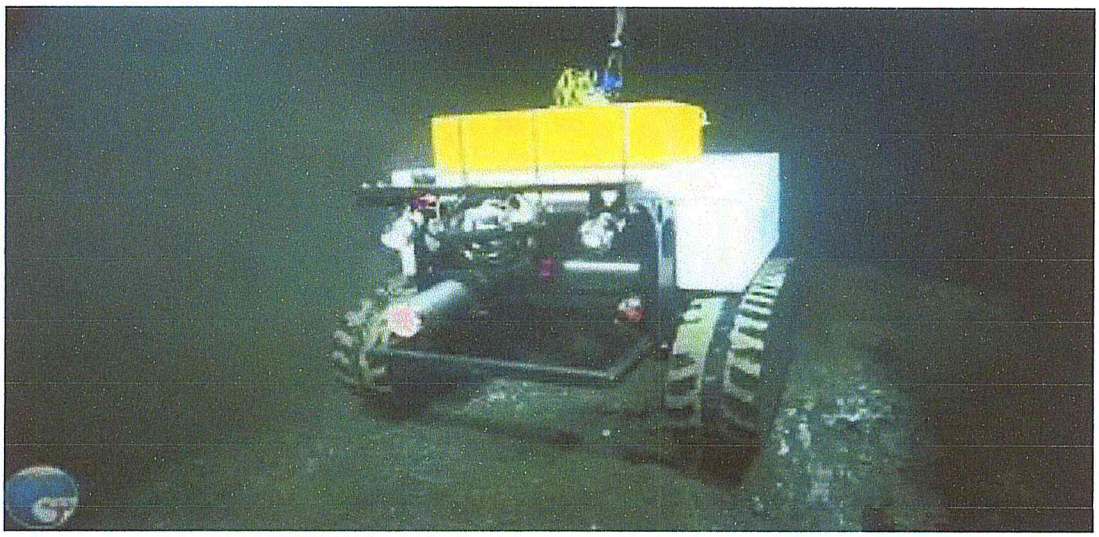


Figure 1.3-- le robot sous-marin ROV[14]

Robotique spatiale : Dans le cadre de la mission mars exploitation rover, les robots spirit et opportunity parcourent mars pour transmettre les informations obtenus grâce à leur capteur vers la terre. L'autonomie d'un robot d'exploitation spatiale est obligatoire, et doit être d'autant plus grande qu'il est éloigné de la terre, du fait du temps qu'il s'écoule entre l'envoi d'une commande depuis la terre, et la réception de cette commande par le robot.

Celui-ci doit donc être capable de réagir tout seul aux évènements qui peuvent surgir dans cet intervalle de temps (voir figure I.4).

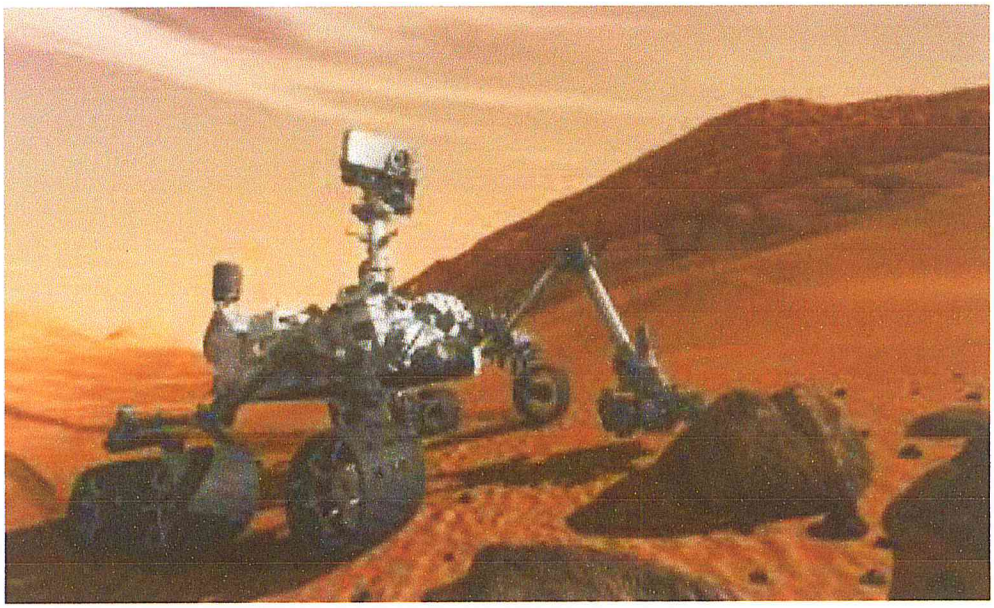


Figure 1.4-- le robot spatial Curiosity[14].

La robotique en milieu hostile : qui intervient en particulier dans le domaine nucléaire pour la surveillance et la maintenance des centrales (reconnaissance et cartographie des niveaux de radiation) [14].

Applications militaires : Un robot militaire est un robot autonome contrôlé à distance conçu pour des applications militaires. On peut citer les drones qui sont une sous-classe des robots militaires. Des systèmes sont déjà actuellement en service dans un certain nombre de forces armées avec des succès remarquables (**voir Figure I.5**).

Figure 1.5-- le robot démineur [15].

I.8 Conclusion

Le monde de la robotique est vaste et complexe car la fonction principal d'un robot et d'aider ou débarrasser l'homme des taches complexes qui se trouve dans le quotidien de son environnement, nous nous sommes focaliser sur la famille robot manipulateur mobile laquelle appartient le robot Robuter/ULM.

Ce premier chapitre constitue une vue générale des notions liées au robot manipulateurs mobiles ainsi que leur domaines d'application. .

Chapitre II

Système de control et middleware robotique

II SYSTEME DE CONTROL ET MIDDLEWARE ROBOTIQUE

II.1 Introduction

Ce chapitre comporte deux grande sections concernant les robots, la première définie les architectures de contrôle et ses différents composant, la deuxième section décrit les middlewares robotiques ensuite elle effectue une étude comparative entre quelque middleware.

II.2 Les architecture de contrôl de mouvement robotique

Dans les entreprises manufacturières, des tâches pénibles, répétitives réalisées par des opérateurs humains peuvent être avantageusement confiées à des systèmes mécaniques articulés dont la dextérité sans égaler celle de l'homme, suffisamment proche de celui-ci pour exécuter des mouvements complexes à l'image de ceux d'un bras humain. Certaines opérations reposent sur l'utilisation des bras articulés. Ces derniers sont des exemples typiques des systèmes soumis uniquement à des contraintes géométriques, les contraintes sur les mouvements viennent de l'évitement des obstacles et des butées articulaires qui limitent les mouvements des bras. Il s'agit de contraintes purement géométriques puisqu'elles ne portent que sur les configurations des systèmes, et non sur leurs dérivées. Des bras articulés sont utilisés pour des tâches répétitives ou pénibles pour un opérateur humain. C'est le cas dans l'industrie automobile par exemple.

II.2.1 Conditions requises pour le déplacement adéquat des robots manipulateurs mobiles

Un robot est susceptible d'évoluer dans un environnement vaste non contrôlé (présence d'obstacles). Cela signifie que les objets peuvent se déplacer, apparaître ou disparaître. L'ensemble des situations ne peut pas être prévu par avance. Le robot devra être muni de capteurs lui permettant d'acquérir des informations sur son environnement proche (caméras vidéo, télémètres ultrasonique ou infrarouge, etc.)

II.2.2 Perception et modélisation de l'environnement

Le robot doit être muni d'un système de perception capable de fournir des informations précises sur l'état de l'environnement qui l'entoure, afin de pouvoir identifier et regrouper des éléments utiles pour une représentation fiable et consistante de cet environnement.

II.2.3 : Les sources d'information

Un système automatique est un système complexe car il nécessite plusieurs sources d'information par les quelles il assure son bon fonctionnement.

II.2.3.i Informations proprioceptives

Les informations proprioceptives renseignent sur le déplacement du robot dans l'espace. Elles constituent donc une source d'information très importante pour la navigation. Cependant, la précision de cette information se dégrade continuellement au cours du temps, la rendant inutilisable comme seule référence à long terme. Cette dégradation continuelle provient de l'intégration temporelle des mesures effectuées par les capteurs internes. En effet, chaque capteur produit une mesure bruitée du déplacement instantané, de la vitesse ou de l'accélération du robot. Ce bruit, via le processus d'intégration qui a pour but d'estimer le déplacement, conduit inévitablement à une erreur croissante.

Malgré ce défaut important, les informations proprioceptives ont l'avantage de dépendre assez peu des conditions environnementales qui perturbent fortement les informations perceptives. La vision, par exemple sera fortement perturbée si l'environnement est plongé dans le noir, mais les informations proprioceptives fourniront une information identique, que l'environnement soit éclairé ou non [15].

II.2.3.ii Informations extéroceptives

Les informations extéroceptives, ou plus simplement les perceptions, fournissent un lien beaucoup plus fort entre le robot et son environnement. En effet, les informations proprioceptives fournissent des informations sur le déplacement du robot, alors que les informations perceptives fournissent des informations directement sur la position du robot dans l'environnement. Ces informations assurent un ancrage dans l'environnement, en permettant de choisir des perceptions qui peuvent être utilisées comme points de repère. Ces points de repère sont indépendants des déplacements du robot et pourront être reconnus quelle que soit l'erreur accumulée par les données proprioceptives. La reconnaissance de ces points est évidemment soumise à une incertitude, mais pas à une erreur cumulative, ce qui les rend utilisables comme référence à long terme [15].

II.2.3.iii l'exploitation de capteurs proprioceptifs et extéroceptifs

L'adjonction de la faculté de vision aux robots, permet d'améliorer de manière significative leurs performances et permet le développement d'applications nouvelles. En effet, les tâches que doivent réaliser les robots peuvent toujours se ramener à des interactions précises avec leur environnement. En l'absence de capteur donnant une mesure directe de la position par rapport aux objets environnants, le robot est aveugle. Dans une telle situation, qui concerne d'ailleurs la majorité des robots, seule une tâche apprise au préalable et répétée dans un environnement connu et invariant peut être envisagée.

La position relative par rapport à l'environnement est alors estimée de manière indirecte en fonction d'une mesure des mouvements du robot. Ainsi, une légère modification de la configuration des objets sur le déroulement de la tâche à accomplir. Par exemple, s'il s'agit de réaliser un travail de soudure au dixième de millimètre près et que l'objet à souder est décalé de un millimètre par rapport à la référence, tous les points de soudure seront décalés de cette même distance. Divers capteurs peuvent être utilisés afin de permettre aux systèmes robotiques de pouvoir appréhender l'environnement. Les plus couramment utilisés sont les capteurs à ultrasons, les télémètres laser, les capteurs d'effort et les caméras. Le capteur à ultrasons a l'avantage de pouvoir être utilisé dans un environnement où la visibilité est faible mais il fournit une information relativement imprécise.

Il est souvent utilisé dans des applications de robotique mobile en association avec d'autres capteurs. Le télémètre laser donne quant à lui une image très précise de l'environnement, mais il nécessite un balayage de l'espace, ce qui est très coûteux en temps. Il est souvent utilisé en extérieur dans des environnements totalement inconnus et difficilement modélisables où il permet de reconstruire la géométrie du terrain même si ce dernier est très peu marqué. Le capteur d'effort fournit également au robot un moyen d'appréhender un peu mieux son environnement. Il est souvent utilisé pour des applications de montage, et plus particulièrement pour des tâches d'insertion. Il permet d'avoir une information sur les forces et les couples d'interaction entre le robot et son environnement.

Ces trois capteurs ont la particularité de réaliser une mesure active sur l'environnement, c'est-à-dire qu'ils fournissent eux-mêmes l'énergie pour la grandeur à mesurer. Une caméra exerce quant à elle une mesure passive, sans interaction avec l'environnement.

C'est le capteur visuel dont les caractéristiques se rapprochent le plus de l'œil humain. Elle permet d'obtenir, en une seule acquisition, une image de l'environnement situé dans son champ de vision. Sur ce point, elle est donc plus rapide que le télémètre laser. Mais, tout comme ce dernier, la caméra est également un capteur précis. La caméra est souvent montée sur l'organe terminal du robot. Mais il arrive qu'elle soit utilisée pour visualiser le robot et son environnement. L'un des buts majeurs de la robotique réside en la création de robots autonomes [7].

II.2.4 Localisation

Le succès dans l'exécution d'une tâche associée à un déplacement est directement lié à la capacité des robots de se positionner par rapport à son environnement. Cette localisation doit être la plus précise possible, et dépend de la fiabilité de la représentation de l'environnement construite par le système de perception du robot.

II.2.5 Exécution de mouvements

Le robot doit être capable de se déplacer de façon sûre à travers l'espace libre de l'environnement, en tenant compte de la présence d'éventuels obstacles statiques et dynamiques.

Le problème de déplacement du robot dans l'environnement rencontre les mêmes difficultés que la localisation et la modélisation liées à la présence d'incertitudes qui font que le déplacement commandé ne sera pas de manière générale exécuté parfaitement. Ces fonctions ne sont pas indépendantes. On note, bien évidemment, que la perception de l'environnement intervient dans toutes.

II.2.6 Déplacement d'un robot manipulateur mobile

Un robot est une machine agissant physiquement sur son environnement en vue d'atteindre un objectif qui lui a été assigné. Cette machine est polyvalente et capable de s'adapter à certaines variations de ses conditions de fonctionnement. Un robot est doté de fonctions de perception, de décision et d'action. Le robot réalise donc de façon continue la boucle (Perception Décision Action), il possède des capacités de mouvement propres et peut entrer en interaction avec des objets de son environnement.

Il a, en outre, la faculté de coopérer à divers degrés avec l'homme. En robotique, la navigation est définie comme une tâche qui consiste, pour le robot, à atteindre un point but dans l'environnement. Le contexte de réalisation de la tâche va conditionner les moyens nécessaires à mettre en œuvre pour permettre au robot de naviguer. Parmi toutes les fonctions requises par un système robotique pour atteindre cet objectif, certaines sont primordiales.

II.2.7 Les architectures de contrôle

Un robot est un système complexe qui doit satisfaire à des exigences variées et parfois contradictoires. Un exemple typique pour un robot mobile manipulateur est l'arbitrage qui doit être fait entre l'exécution la plus précise possible d'un plan préétabli pour atteindre un but et la prise en compte d'éléments imprévus, tels que les obstacles. Ces arbitrages, que ce soit au niveau de l'utilisation des capteurs, des effecteurs ou des ressources de calcul, sont réglés par un ensemble logiciel appelé architecture de contrôle du robot. Cette architecture permet donc d'organiser les relations entre les trois grandes fonctions qui sont la perception, la décision et l'action [17].

Chaque robot à une architecture bien précise, ces architectures peuvent néanmoins être classées en trois grandes catégories que nous détaillerons par la suite : les contrôleurs hiérarchiques, les contrôleurs réactifs et les contrôleurs hybrides. Par contre, toutes ces architectures ne diffèrent pas forcement par les méthodes élémentaires employées mais plutôt par leur agencement et leurs relations.

De nombreuses architectures de contrôle proposées ont fait l'objet de plusieurs classifications. Il faut noter que dans le cadre de problèmes complexes, le mécanisme de contrôle d'un robot est défini par son architecture qui spécifie comment la génération d'actions est organisée à partir des perceptions sensorielles.

De plus, bien que les informations sensorielles soient volumineuses et bruitées, le robot doit réagir rapidement dans certaines situations.

Il est donc indispensable de mettre au point une architecture performante qui permette de remplir la tâche dévolue au robot en utilisant le matériel à disposition. Les architectures utilisées sont hiérarchiques dont nous allons exposer les principales caractéristiques.

II.2.7.i Architectures hiérarchiques de contrôle

Ces architectures ayant pour but de contrôler un système évoluant dans un environnement réel et dynamique, les contraintes de temps de réponse apparaissent clairement dans la perception de la plupart. Ces architectures fonctionnent selon un cycle rigide de modélisation de l'environnement **(voir figure II.1)**, générer des actions au sein de cette représentation, puis exécution du plan. L'essentiel des problèmes de ces architectures provient de l'utilisation d'un modèle interne central qui est le seul pris en compte pour guider le robot. Il faut noter que les systèmes réels ne sont bien souvent pas basés sur une seule catégorie de hiérarchique mais font appel à plusieurs d'entre elles de manière à exploiter les avantages respectifs.

De plus, ces architectures permettent peu de contrôle sur l'exécution des actions. Il n'y a pas de retour direct de la perception sur l'exécution de l'action. Les écarts modèles/environnement ne peuvent être pris en compte que via un nouveau cycle perception/modélisation/planification, ce qui, par définition, est très peu réactif et conduit rapidement à de graves problèmes. La fréquence d'exécution des modules décroît au fur et à mesure que l'on monte dans la hiérarchie. Traditionnellement, le niveau le plus bas (et donc le plus rapide) est chargé du contrôle des moteurs. Le niveau le plus haut (le plus lent) est chargé de la génération de mouvement. Afin d'éviter tout problème d'instabilité, la différence de fréquence entre chaque niveau doit être importante [17].

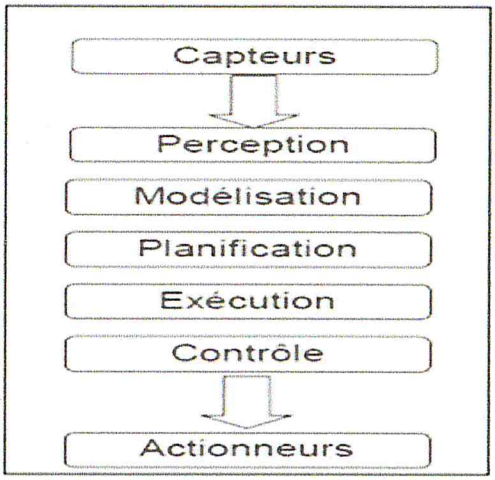


Figure II.1 : Architecture traditionnelle de décomposition du programme de contrôle du robot en différents modules de fonctionnement [17]

II.2.7.ii Contrôleurs réactifs

Rodney Brooks [17] a proposé une solution radicale à tous ces problèmes sous la forme d'une architecture réactive. Dans cette architecture, un ensemble de comportements réactifs, fonctionnant en parallèle, contrôle le robot sans utiliser de modèle du monde.

Cette architecture supprime évidemment les problèmes dus aux différences entre la réalité, d'une part, et le modèle de l'environnement du robot, d'autre part, mais limite clairement les tâches que peut effectuer le robot. En effet, sans représentation interne de l'état de l'environnement, il est très difficile de planifier une suite d'actions en fonction d'un but à atteindre.

Les robots utilisant cette architecture sont donc en général efficaces pour la tâche précise pour laquelle ils ont été conçus, dans l'environnement pour lequel ils ont été prévus, mais sont souvent difficiles à adapter à une tâche différente.

Les réussites de ces architectures sont liées au couplage direct entre la perception et l'action qui permet une prise en compte très rapide des phénomènes dynamiques de l'environnement. Et donc une bonne robustesse dans des environnements complexes. Ces architectures sont en général basées sur plusieurs comportements : évitement d'obstacles, déplacement aléatoire, déplacement vers un but, fuite d'un point. Pour guider le robot, il faut donc choisir à chaque instant lequel de ces comportements activer [17].

Ce problème est connu dans la littérature scientifique sous le nom de sélection de l'action. La solution proposée par Brooks, l'architecture de subsomption est devenue classique et utilise une hiérarchie des comportements qui se déclenchent donc selon un ordre de priorité en fonction des perceptions du robot. En 1986, Rodney A. Brooks propose une approche différente. Cette architecture, appelée subsomption, consiste à paralléliser les tâches. La figure **(II.2)** .

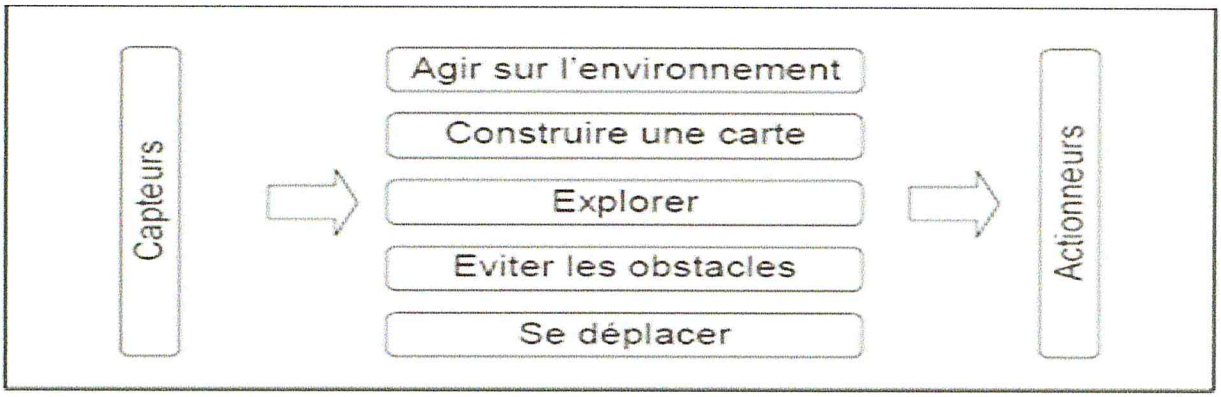


Figure II.2 : Décomposition basée sur le comportement d'accomplissement de tâches [17].

Chacune de ces couches relie les capteurs aux actionneurs et permet un comportement particulier ou une compétence spécifique, comme la locomotion, l'évitement d'obstacles ou la saisie d'objets. Ce type d'architecture permet notamment la décomposition d'une tâche complexe en plusieurs comportements réactifs. Ce type d'approche vise aussi à accroître la fiabilité du système.

Dans l'architecture traditionnelle, si une panne survient sur l'un des modules, alors la panne se généralise à l'ensemble du système, chaque module étant essentiel au fonctionnement de l'ensemble. Dans l'architecture proposée par Rodney A. Brooks, même après la perte d'un module, le système peut continuer à fonctionner en mode dégradé, en inhibant ce module par exemple.

II.2.7.iii Architecture de forçage / inhibition

La figure **(II.3)** représente un module d'une telle architecture. Ses entrées peuvent être forcées pendant une certaine durée, de même les sorties peuvent être inhibées pendant un laps de temps. Des comportements plus complexes peuvent ainsi être construits en imbriquant les modules de base. La structure globale permettra d'inhiber les comportements non prioritaires et de forcer les fonctions vitales. Par exemple le signal d'entrée du module dédié à la locomotion peut être forcé pour le remplacer par un signal d'évitement d'obstacle afin d'empêcher une collision [17].

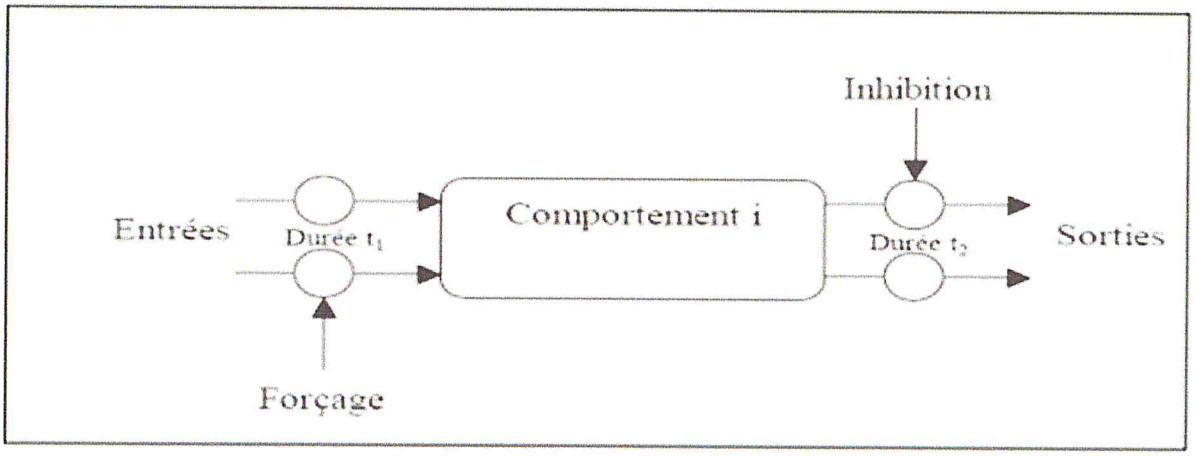


Figure II.3 : Un module d'architecture forçage / inhibition[17]

II.2.8 Contrôleurs hybrides

La plupart des contrôleurs actuellement utilisés choisissent une solution intermédiaire entre ces deux approches sous la forme d'une architecture hybride. Cette architecture se compose de deux niveaux. Le premier est chargé des tâches de navigation de haut niveau, telles que la localisation, la cartographie et la planification.

Pour cela, il s'appuie sur un second niveau réactif qui est chargé d'exécuter les commandes avec le plus de précision possible et de gérer les éléments non modélisés de l'environnement tels que les obstacles inconnus ou dynamiques. L'action conjointe de ces deux niveaux permet de réagir rapidement face aux variations imprévues de l'environnement, tout en permettant la réalisation d'actions planifiées à plus long terme. Le bas niveau de ces architectures peut être réalisé sous forme de comportements, tels que ceux utilisés dans les architectures réactives. Ces comportements sont des boucles sensorimotrices qui relient les actions aux perceptions avec une phase de décision très courte, qui assurent la réactivité. Dans le même temps, les informations sensorielles sont utilisées par le haut niveau dans une boucle sensorimotrice à une échelle de temps beaucoup plus longue. C'est la mise en parallèles de ces deux échelles de temps qui fait la force de ces architectures.

II.2.9 Architecture de control pour le robot mobile SPINOS

La figure II.4 présente la structure du logiciel de contrôle du robot SpinoS. On y retrouve principalement deux sections, soit le module temps réel et le programme de contrôle. Les prochaines pages décrivent chacun des éléments de cette structure, ainsi que les entrées/sorties entre chaque élément. Il est à noter que cette structure présente strictement ce qui est propre au robot SpinoS et ne donne aucun détail sur les modules de bas niveau qui permettent de communiquer avec le matériel. En fait, deux modules supplémentaires sont présentement utilisés, soit le module de communication avec la carte de contrôle (pmd.o) et le module de communication avec la carte d'entrées/sorties numériques (pcm3724.o). Le module temps réel doit évidemment utiliser les modules de communication avec le matériel. Les modules développés par l'équipe se trouvent tous dans src/sae/modules[36].

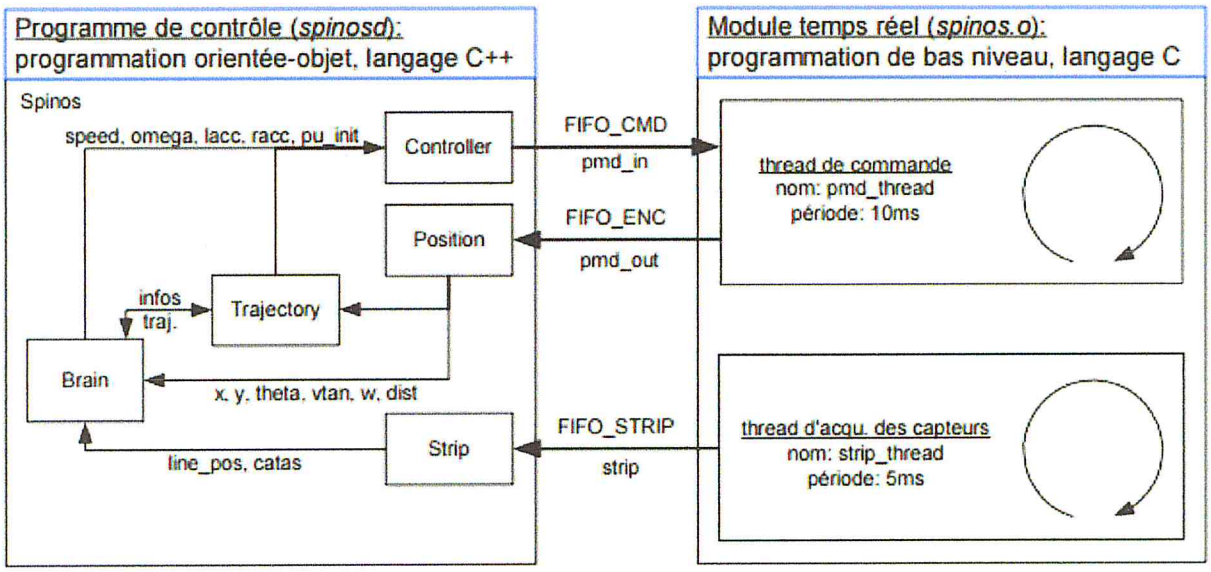


Figure II.4 : la structure du logiciel de contrôle du robot SpinoS[36].

II.2.10 La commande référencée vision/architecture de control d'un robot manipulateur

Définition

Par définition, *la commande référencée vision* ou *asservissement visuel* AV (*Visual servoing* en englais) consiste à contrôler les mouvements d'un système robotique en intégrant les informations fournies par un capteur de vision dans un système de commande en boucle fermée.

La figure II.5 expose le schéma classique de la commande incluant les différents maillons de la chaine de traitement :

*s**: le vecteur de *primitives visuelles de références* qui est obtenu à partir de l'image de référence.

s: le *vecteur des primitives courantes* obtenu à partir de l'acquisition de l'image courante. *e=s-s**: erreur visuelle est la différence entre le vecteur de primitives courantes et le vecteur des primitives de références.

vc: le signal de commande ou la vitesse de référence de déplacement de la caméra qui est envoyé au *système robotique* dans le but de déplacer son effecteur c'est-à-dire avoir *une nouvelle situation de la caméra.*

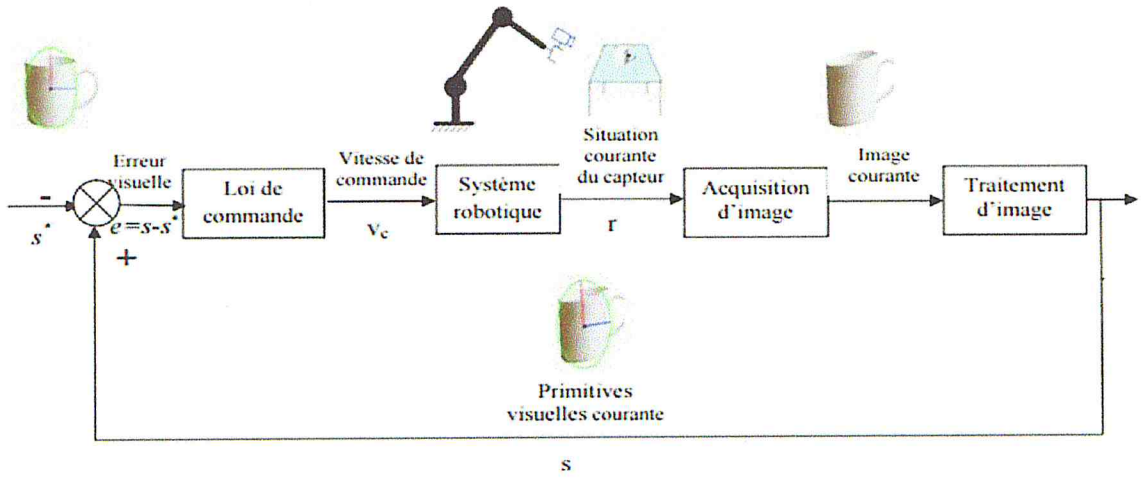


Figure II.5 : Schéma de commande par asservissement visuel[37]

Après l'acquisition de la nouvelle image, elle va passer par *un traitement* dans le but d'extraire des nouvelles primitives.

En ce qui concerne le capteur de vision, les premiers travaux concernant l'utilisation d'un capteur de vision pour la commande des robots manipulateurs remontent dans les années

70. C'est à Hill et Park [37] que l'on doit l'apparition du terme asservissement visuel (*visual servoing).*

Il implique la mise en œuvre d'au moins 3 sous-systèmes :

- Un capteur visuel qui est une caméra.
- Un dispositif d'acquisition/traitement, autrement dit un ordinateur, qui est chargé du traitement d'images et de la commande.

- Un sous-système mécanique actionné représentant le robot qui est chargé de l'exécution de cette commande. Le robot génère ses mouvements de façon à ce que sa cible visuelle atteigne une certaine configuration dans l'image qu'il perçoit (il faut assurer que l'objet reste dans le champ visuel de la caméra pour qu'il soit toujours visible pendant l'exécution de la tâche).

L'asservissement visuel prend en considération plusieurs domaines de recherche, y compris la modélisation du robot (géométrique, cinématique, dynamique), la commande des systèmes en temps réel, la vision par ordinateur (l'extraction des informations visuelles, calibration de la caméra). Il touche aussi:

- Le domaine aérien : les applications les plus courantes sont la surveillance civile avec le suivi de structure de linéique (routes, lignes électriques), l'assistance au pilotage (hélicoptère), appontage automatique d'avions, les applications militaires (drones).
- Le domaine medical.
- La conduite d'un véhicule sous-marin.

II.3 Les middlewares robotique

Il existe une variabilité dans les concepts qui composent une application de robotique. Notamment la variabilité du matériel, a orienté la conception de certains middleware de robotique pour simplifier le processus de développement de ces applications rendre ces applications indépendantes des détails du matériel assurer la connexion entre les différents modules d'une application distribuée.

II.3.1 Définition

De l'architecture logicielle en robotique : Un Middleware (intergiciel) est un logiciel tiers qui créé un réseau d'échanges d'informations entre différentes applications informatiques. Il permet de diviser la partie logicielle du robot en une suite de processus pouvant être démarrés simultanément ou séquentiellement au lancement d'une mission [18].

II.3.2 L'intérêt d'un middleware :

C'est pouvoir :

- séparer les applications et donc clarifier la répartition du travail au sein d'une équipe.
- paralléliser facilement les processus et tirer profit des n cœurs du processeur utilisé.

- distribuer les applications sur différentes machines car elles sont toutes liées à un serveur.
- rejouer des missions car il est facile de logger tous les changements de variables.
- réutiliser les applications déjà implémentées qui répondent à nos problématiques.

II.3.3 Les Intergiciels existants

Dans cette section, nous présentons les intergiciels les plus utilisés, nous avons choisi 5 intergiciels pour la robotique : Player/Stage, ROS, Miro, MRDS et Marie.

Player/Stage [18] **:** Est conçu pour fournir une infrastructure, des pilotes et une collection de bibliothèques de périphériques partagés pour les applications robotiques. C'est l'un des premiers intergiciels qui a émergé pour les systèmes robotiques, et d'autres intergiciels utilisent Player comme fondation.

Player/Stage ne considère pas un robot comme une unité, mais traite séparément les dispositifs, ce qui en fait un serveur de référentiel pour actionneurs, capteurs et robots.
Les principales caractéristiques de Player sont le serveur de référentiel des périphériques, la variété des langages de programmation, le protocole de transport basé sur des sockets, la modularité et le fait d'être open-source.

Cet intergiciel est composé de deux éléments : Player et Stage. Player représente l'intergiciel lui-même et Stage est un simulateur 2D.

ROS [19] **:** (Robot operating system) est un intergiciel récent et souple pour les applications robotiques. C'est une collection d'outils, de bibliothèques et de conventions qui visent à simplifier la dé- finition de comportements complexes et robustes pour une grande variété de plateformes robotiques. Il fournit une abstraction du matériel, des pilotes de périphériques, des visualiseurs, une infrastructure pour communiquer à travers des messages et une gestion des paquets. ROS est livré avec une série de bibliothèques contenant des services robotiques comme des fonctions de SLAM 3, de navigation autonome, ou de suivi d'objets. ROS est conçu pour être multiplateforme.

Miro [20]**:** est un intergiciel orienté objet, distribué, développé pour améliorer le processus de développement de logiciels en augmentant l'intégrabilité des logiciels hétérogènes, la modularité et la portabilité des applications robotiques. Il a été développé en C++ pour Linux basé sur CORBA (Common Object Request Broker Architecture). Cela permet l'interopérabilité entre plateformes, rendant l'intergiciel applicable dans un contexte distribué multi-robot.

Microsoft Robotics Developer Studio [21] : (MRDS) est un intergiciel basé sur Windows pour le contrôle et la simulation du robot créé par Microsoft. Visual Programming Language, qui est un élément clé du MRDS, est un environnement de développement graphique qui utilise un catalogue de service et d'activité.
MRDS vise un public universitaire, amateur et les développeurs commerciaux.

Il gère une grande variété de matériels et robots comme Eddie Robot, ABB Group Robotic, CoroWare CoroBot, Lego Mindstorms NXT, iRobot Create, Parallax Boe- Bot etc.

Marie [22]: (Mobile and Autonomous Robotics Integration Environment) est un intergiciel conçu pour permettre l'intégration et la distribution de logiciels pour les systèmes robotiques.

Il a été créé en C++ et utilise Adaptive Communication Environment (ACE) comme infrastructure de communication. Le composant central fourni par l'intergiciel, appelé Mediator Design Pattern (MDP), permet aux composants logiciels de se connecter à MARIE. MARIE peut fonctionner sur MobileRobots Pioneer 2. Ses principales caractéristiques sont l'interopérabilité et la réutilisation des modules logiciels robotiques.

II.3.4 Les critères comparatifs

Nous allons comparer les cinq intergiciels robotiques présentés ci-dessus d'un point de vue génie logiciel, car le concept d'intergiciel est apparu dans ce domaine et qu'il fournit de nombreuses caractéristiques qui peuvent être transférées aux applications robotiques. Nous avons regroupé les critères de comparaison en trois grands groupes : architecture, infrastructure et usage [23]. Chaque groupe est composé de différents critères présentés ci-dessous. L'architecture évalue l'impact qu'a l'intergiciel sur le système d'exploitation hôte, qui se décompose en :

Système d'exploitation : la dépendance du système d'exploitation. Ce critère exprime la portabilité d'un système au travers de multiples plateformes et systèmes.

Services de stockage de données durable : des outils qui permettent de conserver les données de capteurs et d'autres robots de la flotte. La couche de persistance des données est importante pour la sauvegarde des résultats de la mission, pour les validations de données expérimentales, pour le traitement de données hors ligne ainsi que pour rejouer les données dans un simulateur.

Tolérance aux pannes : elle correspond à la détection d'une panne de logiciel, d'un fonctionnement en mode dégradé et à l'exécution de processus de récupération. Le fait qu'un intergiciel puisse détecter des défaillances est essentiel pour les applications robotiques. En outre, il est important que les robots continuent d'effectuer leurs tâches dans un mode dégradé jusqu'à ce que le système répare la défaillance.

L'infrastructure évalue les outils et les API fournies par l'intergiciel, qui se décompose en :

Processus de gestion et de surveillance : des outils qui permettent de gérer, de déboguer, de configurer et de surveiller les composants de l'intergiciel.

Il est important de faciliter la tâche de surveillance en offrant une vision complète des capteurs, des actionneurs et du statut des autres robots.

Services de coordination multi-robot : outils pour le partage de données entre robots et de distribution des tâches. Il est important de disposer d'outils de gestion de distribution des algorithmes afin de réduire la complexité du développement.

Communication : La communication est très importante entre les différents composants d'un robot afin de lui permettre d'effectuer avec succès sa tâche, ainsi qu'à l'intérieur d'une flotte de manière à permettre à des robots d'interagir avec les autres.

L'usage évalue l'apport de l'intergiciel pour la définition de nouvelles applications, qui se décompose en :

Déploiement et cycle de vie : la possibilité de déployer des comportements/configurations sur l'ensemble de la flotte, d'intégrer les chaînes de compilation et la gestion du cycle de vie pour de nouvelles applications.

Il est très utile d'avoir un système de compilation et un environnement de test automatisé pour simplifier le déploiement et la définition de tâches distribuées.

Modèle de programmation : les types de programmation disponibles : synchrone 4, asynchrone 5, etc. Le fait d'avoir des approches différentes pour les modèles de programmation permet d'employer simultanément différentes techniques selon les problèmes traités.

Services d'intégration du code et des données : facilite l'intégration de nouveaux services et de modules via des APIs dans le logiciel embarqué. La présence d'interfaces permet au développeur d'enrichir les applications et de faciliter le développement.

II.3.5 Comparaison des intergiciels

Cette section analyse chaque intergiciel selon les critères présentés dans la section précédente.

Chaque grand groupe est commenté dans un paragraphe. L'évaluation des critères est notée comme suit : un + représente que toutes les exigences du critère sont satisfaites, un + représente que la plupart des exigences sont présentes, un ~ montre le fait que le critère n'est satisfait que partiellement et un - représente une absence des exigences.

II.3.5.i L'Architecture:

Intergiciel	*Système d'exploitation*	*Services de stockage de données durable*	*Tolérance aux pannes*
Player/Stage	Linux, Windows	~	+
ROS	Des dépôts: Ubuntu, Debian. From source : generic Linux. Windows, MacOS	+ (Rosbags)	+
Miro	Linux	~	+
MRDS	Windows	~	+
Marie	Linux	~	+

Tableau II.1—L' Architecture

Le Tableau **(II.1)** synthétise le groupe Architecture. Il est composé des critères : système d'exploitation, services de stockage de données durable et tolérance aux pannes. Système d'exploitation.

Services de stockage de données durables : ROS est le seul intergiciel qui fournit des services de stockage de données durables. Tous les messages produits peuvent être conservées dans le nœud rosbags. Les autres intergiciels ne fournissent aucune API native pour enregistrer les informations des capteurs.

Tolérance aux pannes : Aucun intergiciel ne dispose d'un mode dégradé ni d'un système de redémarrage automatique des processus après un échec.

ROS a besoin d'une adresse IP à l'initialisation pour exécuter le nœud de démarrage roscore. Une fois que tous les nœuds sont lancés, l'échec de roscore n'affectera pas les autres nœuds.

II.3.5.ii L'infrastructure

L'Intergiciel	*Processus de gestion et de surveillance*	*Services de coordination multi-robot*	*Communication*
Player/Stage	~	+ (Coordination par tiers)	~
ROS	+(Gestion et surveillance)	~	+ (Sync.& asyne.)
Miro	- (Aucun)	~	~
MRDS	+ (Visual Studio plugins)	~	+ (Sync.& async.)
Marie	~	~	~

Tableau II.2 : L'infrastructure

Le Tableau **(II.2)** résume le groupe Infrastructure **:** Il est composé de : processus de gestion et de surveillance, services de coordination multi-robot et communication.

Processus de gestion et de surveillance : À part Miro qui ne fournit ni une surveillance ni une interface de gestion, les autres intergiciels disposent d'un logiciel de surveillance graphique. MRDS utilise Visual Studio comme IDE. ROS a plusieurs outils de gestion et un tableau de bord graphique QT.

Services de coordination multi-robot : Aucun des intergiciels considérés ne fournit de services natifs de coordination multi-robot. Player/Stage dispose d'algorithmes de coordination externe développés pour lui. ROS, Miro, MRDS, Marie et délèguent les services de coordination à la couche d'application.

Communication : La communication entre les couches de l'infrastructure dans Player/ Stage est faite en utilisant des connexions socket directes. Le partage des données dans Miro est affecté à la IIOP 7 de CORBA. Marie utilise une mémoire partagée et des sockets. MRDS et ROS disposent de communications à la fois synchrones et asynchrones.

II.3.5.iii Usage

Le Tableau (II.3) résume le groupe Usage, Il est composé de : déploiement et cycle de vie, modèle de programmation et services d'intégration du code et des données.

Déploiement et cycle de vie : Aucun des intergiciels ne fournit un système de déploiement multi robots. ROS ne possède pas de système de référentiel de déploiement mais dispose d'une chaîne de compilation à base de CMake appelé Catkin. Il utilise le simulateur Gazeboo comme environnement de test. Miro a un compilateur IDL, qui permet de générer le code pour la communication entre l'intergiciel et les services sous adjacents.
Il utilise Stage et Gazeboo pour les simulations. MRDS utilise Visual Studio comme IDE qui fournit une chaîne de compilation, un outil de déploiement, ainsi qu'un simulateur. Player/Stage n'a pas de chaîne de compilation native mais il existe des chaînes de compilation dans les IDE pour compiler le code source de l'application. Il fournit un environnement de test dans Stage. Marie n'a pas d'outil de compilation spécifique ou de système de déploiement.

Modèle de programmation : Les applications pour Player/Stage peuvent être écrites dans n'importe quel langage de programmation. Les applications Miro peuvent être écrites dans tous les langages fournissant des implémentations de CORBA. Les échanges de données sont déclenchés par un événement. MRDS utilise un langage de programmations visuelles, un environnement de développement graphique qui utilise un catalogue de services et d'activités. Le langage de programmation principal dans MRDS est C#. ROS supporte à la fois des modèles de programmation synchrones et asynchrones.

Les applications peuvent être écrites nativement en Python et C++, mais il existe des intégrations pour Java, LISP et d'autres langages. ROS reçoit la meilleure évaluation en raison de la variété des langages et des modèles de programmation dont il dispose.

Services d'intégration du code et des données : Tous les intergiciels examinés supportent une architecture modulaire et permettent une intégration facile ou une réutilisation du code.

Miro fournit des abstractions de service des capteurs et des actionneurs via le langage de définition d'interface (IDL) CORBA. MRDS, avec l'utilisation de VPL 8, permet de générer le code de nouvelles "macro" de services à partir des diagrammes créés par les utilisateurs.

Un service ou une activité est représentée par un bloc avec des entrées et des sorties qui a juste besoin d'être glissé du catalogue au diagramme. Marie fournit des services de traduction tels que les composants écrits pour Player / Stage peuvent être utilisé. ROS a un système de packages bien conçu et un système du lancement gérant les dépendances. En synthèse des tableaux établis ci-dessus, il apparaît que ROS semble être l'intergiciel le plus approprié pour les systèmes multi robots, suivis par MRDS.

Les deux satisfont totalement ou presque les différents critères proposés. Dans la section suivante, nous pré- sentons une autre approche émergente, le cloud pour les robots.

L'Intergiciel	*Déploiement et cycle de vie*	*Modèle de programmation*	*Services d'intégration du code et des données*
Player/Stage	~	~	+
ROS	+(Catkin)	+ (Async et sync)	+ (roslaunch. rosrun)
Miro	- (IDL compiler.Gazeboo)	- (CORBA)	~
MRDS	+ (Visual Studio)	+(C#)	+ (VPL)
Marie	- (Aucun)	~	+ (compatible Player)

Tableau II.3 :--L' Usage

D'après cette comparaison on a utilisé le middleware ROS.

II.4 Conclusion

Dans le domaine de la robotique, le control du robot est liées directement à sa couche logicielle dont le choix des outils et la configuration est primordiale pour le bon déroulement de l'exécution.

Dans ce chapitre on a présenté les différent architecture de control de robot ainsi que leurs outils et exigence de fonctionnement comment la perception de l'environnement à partir du capteur et la modélisation de ces données pour la exploitation.

On va exposée également le middleware robotique les plus connus ainsi que leur caractéristiques et les fonctionnalités requises, ensuite effectué une comparaison entre ces middlewares selon certaines critères.

Chapitre III

Conception de l'architecture

III CONCEPTION DE L'ARCHITECTURE

III.1 Introduction

Le développement d'un robot autonome pose de nombreux problèmes fondamentaux dans des domaines variés et bien distincts. Le problème posé est de déterminer à chaque instant quelle commande doit être envoyée aux effecteurs, connaissant d'une part le but à accomplir et d'autre part les valeurs retournées par les différents capteurs.

Il s'agit de déterminer les liens existants entre la perception et l'action connaissant les buts à atteindre. Nous nous intéressons plus particulièrement au cours de cette étude au choix des actions permettant au robot d'atteindre un point de l'environnement spécifié par ses coordonnées, tout en évitant les différents obstacles présents.

III.2 A propos du robot

Le RobuTER consiste une plateforme mobile sur laquelle est monté un bras manipulateur.

III.2.1 Plateforme mobile

Le RobuTER est une plate-forme (**voir FigureIV.1**) automatisée et programmable de transport d'objets de taille moyenne (jusqu'à 150kg). Ses deux roues motrices permettent une grande mobilité sur sols lisses.

Grâce à son pilotage en différentiel de vitesse, le RobuTER peut tourner sur lui-même. Dans la version proposée, il est équipé d'un bras Ultra-Léger (**voir FigureIV.2**).

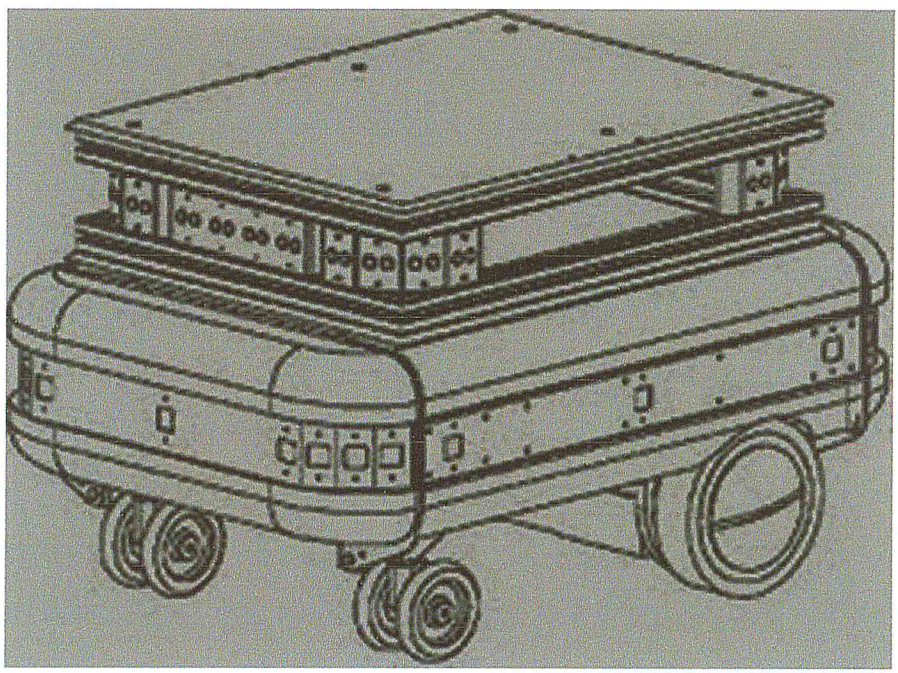


Figure III.1-- Vue du RobuTER[35]

III.2.2 Le Bras Ultra Léger :

Pesant 20kg, le bras Ultra-Léger est un manipulateur peu onéreux, prêt à l'emploi, pouvant être utilisé sur une plateforme mobile.

Il est aussi bien adapté pour le transport ou la manipulation de petites pièces, que le ramassage d'échantillon au sol.

L'ensemble de la plate-forme munie du bras est contrôlé par un PC embarqué (Pentium MMX233 MHz sous Linux) et 3 contrôleurs cb555 fabriqués par ROBOSOFT. Le premier cb555 [35] se situe à l'intérieur de la base mobile, tandis que les 2 autres sont installés dans le contrôleur du bras. Cette architecture de contrôle présente l'avantage pour les utilisateurs de programmer les mouvements de la plate-forme soit en utilisant les outils de développement orienté « haut niveau », soit en utilisant la chaîne de compilation dédiée SynDEx pour l'accès au « bas niveau ».

Les caractéristiques de l'ensemble sont les suivantes :

- 2 roues motrices à bandage de polyuréthane
- Direction par différentiel de vitesse
- Bras manipulateur à 6 axes avec pince

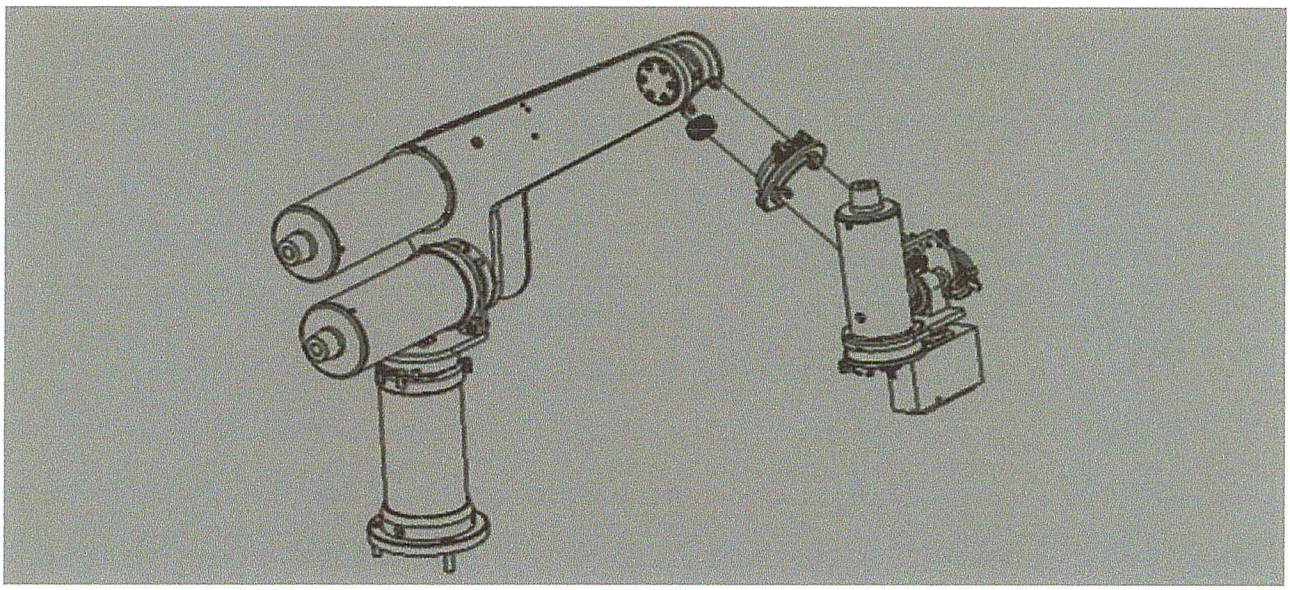


Figure III.2--Vue du Bras Ultra Léger[35]

III.2.3 Système de mouvement de RobuTER :

Bien que ce système permet d'assembler différents composants, ils ne proposent pas de moyens de les contrôler finement, i.e. de configurer, d'ordonnancer, de déclencher et suivre l'exécution de ces différents composants.

Cette notion d'assemblage et de contrôle est primordiale pour la bonne réalisation de missions autonomes. De nombreuses solutions ont été proposées pour le domaine de la robotique sous divers noms ?architecture hiérarchique, architecture réactif, architecture hybride . . . il s'agit principalement de la tâche de coordination, avec en filigrane une partie configuration, souvent relativement statique.

III.3 Exigences pour une architecture de contrôle

Une architecture de contrôle a donc pour vocation de coordonner les différents éléments de la couche fonctionnelle afin d'effectuer différents types de missions, tout en ayant la capacité de réagir efficacement à différents types d'évènements, le tout dans un monde difficilement prédictible **[33]**. Pour cela, elle doit considérer et traiter les problématiques suivantes :

- **Réaction et délibération** : Un robot autonome évolue dans un environnement complexe et dynamique et doit donc réagir en conséquence. Dans le même temps, pour réussir des missions nécessitant des comportements complexes, il doit être capable de considérer des modèles et des plans à plus long terme. L'architecture doit donc permettre de prendre en compte ces différents niveaux de raisonnement.

- **Gestion de la concurrence :** Un robot autonome est un système hautement concurrent, où de nombreuses tâches parallèles vont potentiellement accéder et/ou modifier un nombre fini de ressources, tant logicielles que matérielles. Le robot, au travers de l'architecture de contrôle doit être capable de raisonner sur son état global courant et sur la mission en cours afin d'assurer un comportement efficace et cohérent, en particulier en évitant les conflits sur ces différentes ressources :

– **Robustesse :** La majorité des fonctionnalités à bord d'un robot peuvent échouer (en particulier l'architecture de contrôle), soit pour des raisons internes (erreurs dans l'implémentation d'un algorithme, algorithme inapproprié pour la situation, échec d'un des capteurs ou actionneurs, . . .), soit pour des raisons liées à l'environnement physique du robot.

Toutefois, dans tous les cas, l'échec d'un des composants ne doit pas entraîner l'échec de la mission courante : l'architecture doit donc être capable de détecter et de raisonner sur les différentes causes possibles d'échecs, et si possible de trouver une solution alternative pour achever la mission.

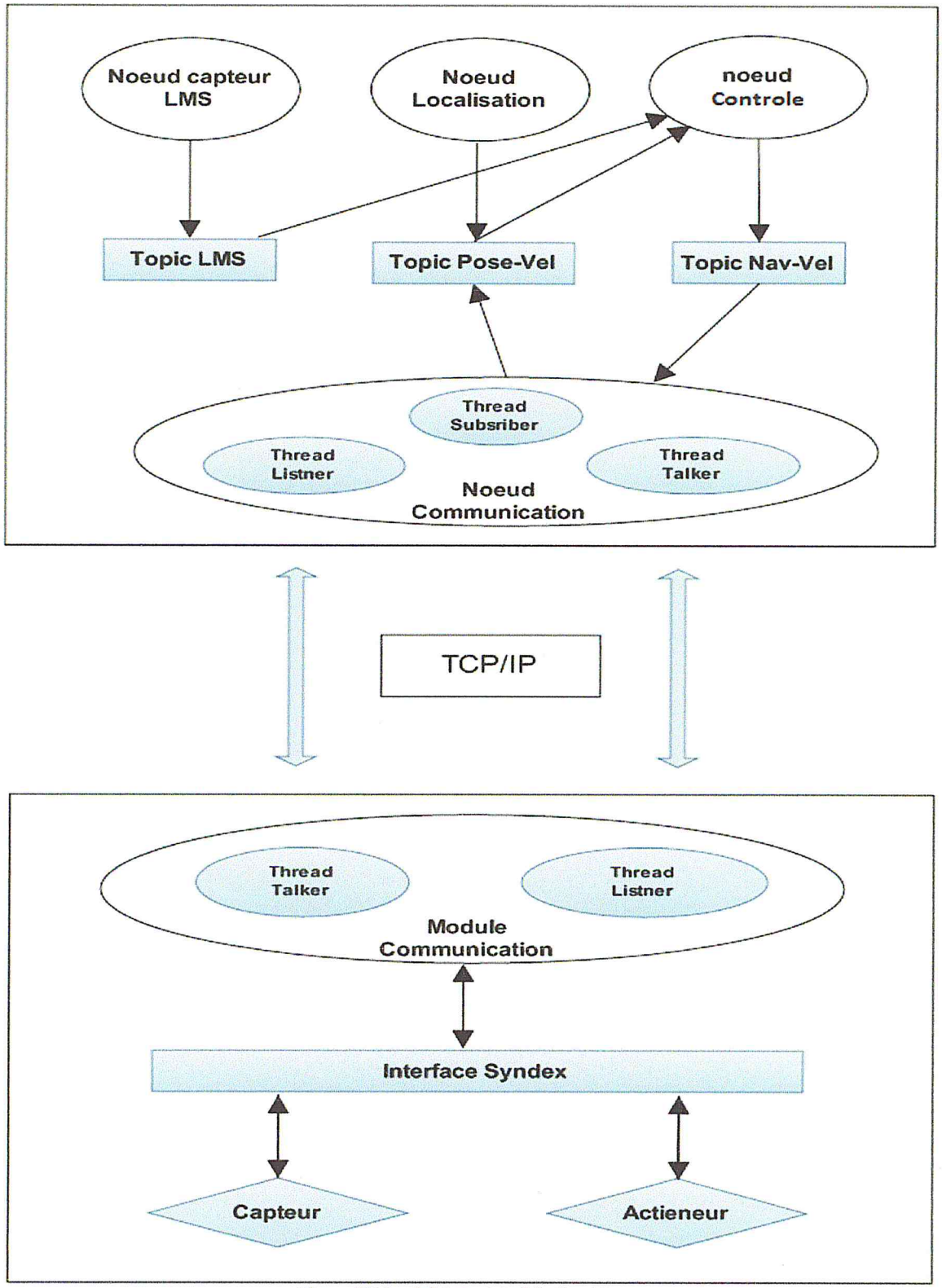
– **Vérification :** Un robot autonome est un système complexe, agissant dans un monde complexe. Pour garantir la sécurité des hommes interagissant avec le robot, ou bien l'intégrité du robot, il est nécessaire de pouvoir certifier certaines propriétés du robot telle que "le robot va-t-il toujours s'arrêter avant de heurter un humain ?". Traditionnellement, les développeurs utilisent des tests pour se convaincre que de telles propriétés sont vérifiées par le système, mais les tests peuvent difficilement couvrir tous les cas possibles dans un environnement complexe et dynamique. Ils ne peuvent donc qu'augmenter la confiance dans le système, pas le garantir. Il est donc important qu'une telle architecture soit modélisable afin de lui appliquer des méthodes formelles de vérification.

– **Modularité et composition** : Le développement d'un robot versatile est souvent un processus long et itératif et dépendant de nombreux acteurs. Il est donc primordial d'avoir une architecture aussi modulaire et composable que possible : l'introduction de nouveaux comportements ne doit, dans la mesure du possible, ni casser les assemblages existants, ni demander de récriture majeure de ceux-ci. De plus, il doit être possible d'ajouter ou de supprimer dynamiquement des fonctionnalités. Ces propriétés existent au niveau de la couche fonctionnelle et doivent être étendues à l'architecture de contrôle.

Au final, le but d'une architecture de contrôle est donc de faciliter le travail des développeurs en leur proposant un cadre de travail assez expressif pour définir des schémas de contrôle et les stratégies de récupérations spécifiques à certaines fautes, tout en étant capable de raisonner sur elle-même, afin de réparer, de manière la plus automatique possible, les erreurs liées à la concurrence ou à une faute du système.

III.4 Architecture de perception et de génération de mouvement

D'après notre étude sur l'environnement existant et les exigences prescrit précédemment cités on a proposé l'architecture représenté sur la figure IV.3 :



FigureIII.3 --Schéma global de l'architecture

Afin de faciliter le développement d'un tel système, on se propose de diviser le problème en deux grandes sous-parties :

III.4.1 Partie PC_ROS

C'est la partie distante qui est le principal élément commandeur du robot manipulateur mobile. Elle permet de définir la manière dont les différents composants peuvent échanger des informations entre eux. Elle est composées des nœuds suivant.

- **Le nœud Communication :** Il s'agit d'un nœud ROS qui est abonné au topic Nav-Vel pour récupérer les vitesses publiées par le nœud contrôle, et il comporte deux sockets tcp/ip pour communiquer avec la partie embarqué. De plus ce nœud récupéré les vitesses actuelle des roux de la base mobile et des axes du bras manipulateur à partir du pc embarqué puis publie ces derniers dans le topics pose-vel.
- **Le nœud de contrôle** : c'est le nœud qui permet de calculer les vitesses des roues de la base mobile de Robuter et de calculer les vitesses et les positions du bras-manipulateur grâce à des fonctions géométriques. Ce nœud englobe la stratégie de navigation de la plateforme, le mouvement du bras, le contrôle manuel du robot et le mode planification.
- **Le nœud localisation** : ce nœud a pour mission de localiser la position du robot à partir des données renvoyé par la cameras Kinect et les publier dans le topic LOC/LMS.
- **Le nœud capteur LMS** : ce nœud permet de situer la position de l'obstacle dans l'environnement du robot à partir des données récupérées par le capteur laser et les publier dans le topic LOC/LMS.

III.4.2 Partie PC_EMBARQUE

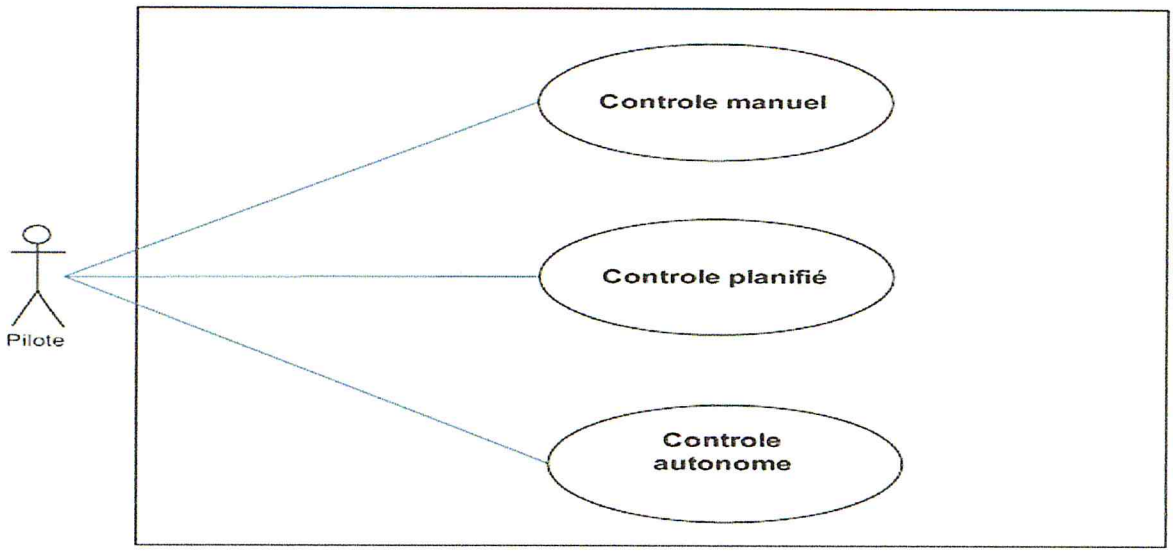
- **Le module Communication :** c'est un programme écrit en C qui comporte deux threads et des fonctions d'appel d'actionneur et de capteurs.
- **Interface syndex :** Une API qui joue le rôle d'intermédiaire entre le programme de communication (langage haut niveau) et les carte électroniques (bas niveau).
- **Actionneurs :** Des moteurs qui font mouvoir les roues de la base mobile ainsi que les axes du bras manipulateur.
- **Capteurs :** Un scanner LMS qui fait partager des signaux laser dans l'environnement du robot et détecte l'obstacle ainsi qu'ne camera Kinect qui se place à l'avant du robot et localise la position du robot.

III.5 Modélisation du système

On a choisi le langage UML pour modéliser le système, et on a utilisé les diagrammes : cas d'utilisations, communication et séquence.

III.5.1 Diagrammes des cas d'utilisation

La figure IV.4 présente un diagramme d'utilisation globale de notre application



FigureIII.4 --: Diagramme de cas d'utilisation globale

Nous allons représenter l'ensemble des cas d'utilisation recensés et l'acteur intervenant dans chacun de ces cas :

- **Control manuel**
- **Control Planifié**
- **Control Autonome**

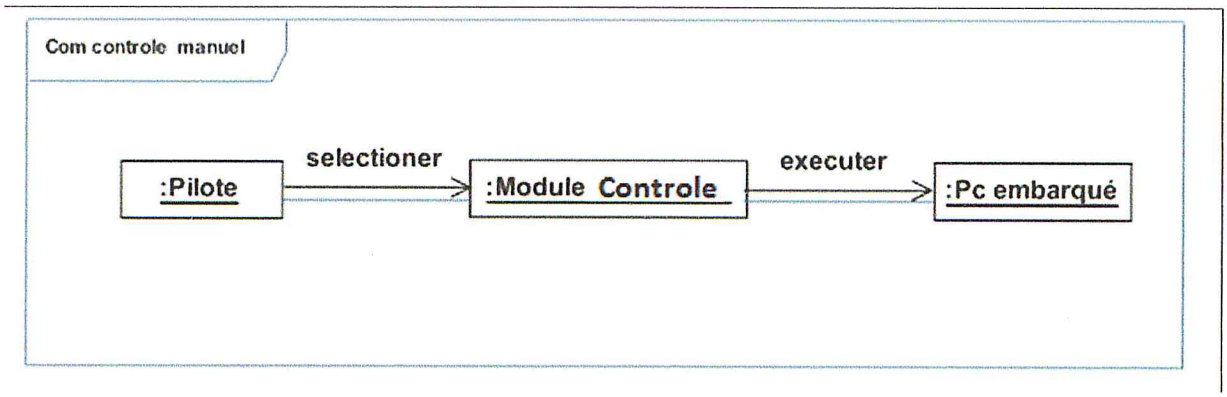
III.5.2 Cas d'utilisation « Contrôle manuel » :

Le but de ce mode est de permettre au pilote de contrôler le robot de manière manuelle sans activer les moyens de manipulation pour des actions primitives.

Afin d'effectuer le pilotage manuel le système offre au pilote la main d'affecter au robot des vitesses pour les roux et les axes du bras manipulateurs

III.5.2.i Diagramme communication « Contrôle Manuel »

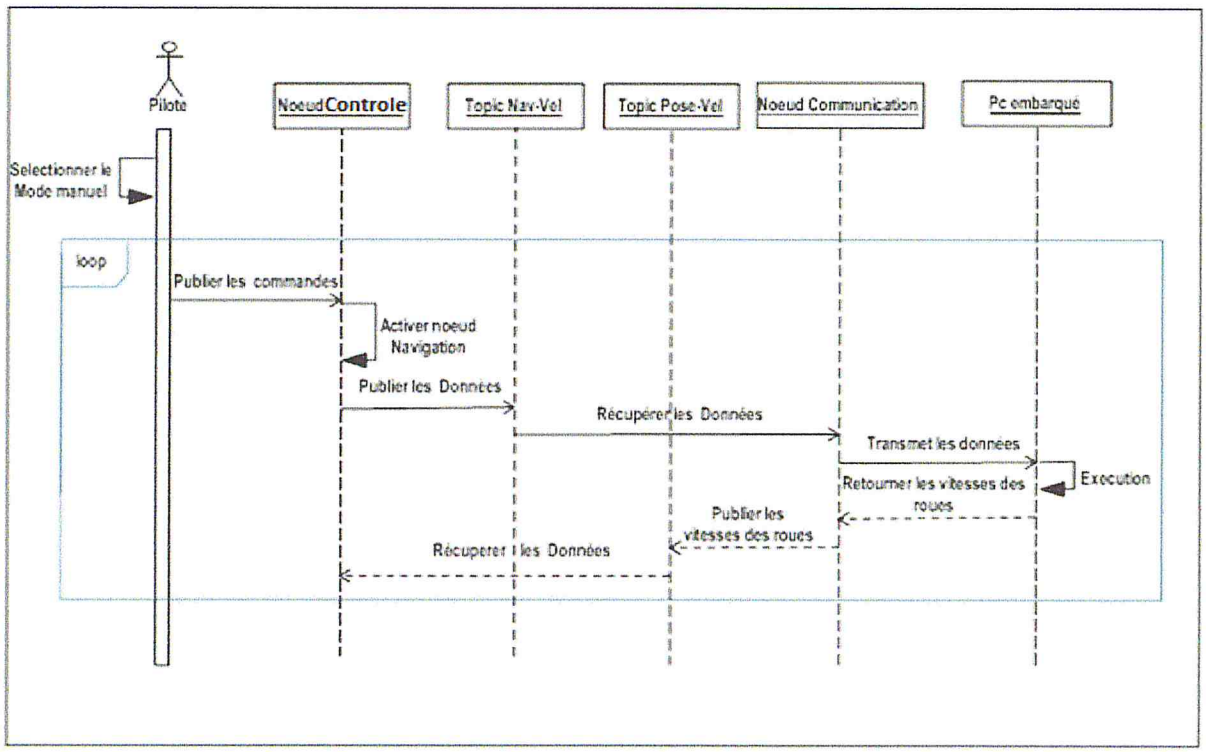
Ce diagramme résume les principaux modules impliqués dans le mode manuel, et les différentes interactions entre eux.



FigureIII.5-- Diagramme de communication du contrôle manuel

III.5.2.ii Diagramme séquence « Pilotage manuel »

Ce diagramme représente les interactions entre le pilote et le système selon un ordre chronologique .



FigureIII.6-- Diagramme de séquence « Contrôle Manuel »

Scénario « Pilotage manuel »

Tout d'abord Le pilote sélectionne le mode de control manuel et saisie les vitesses des roues de la base mobile et des axes du bras manipulateur, ensuite les informations saisie sont envoyées au nœud control. Ce dernier traite les données reçues par le pilote et génére les mouvements nécessaires pour cette requête, après que ces données soient publiées dans le topic nav-vel. Celui-ci est abonné par le nœud communication qui à son tour récupère ces données de ce topic et les transmet au pc embarquée via socket tcp/ip.

La partie embarqué situé dans le pc embarqué Réceptionne les commandes et les affecte aux actionneurs pour l'exécution. Quand le robot commence de mouvoir il revoie les vitesses des roues et du bras manipulateur courants.

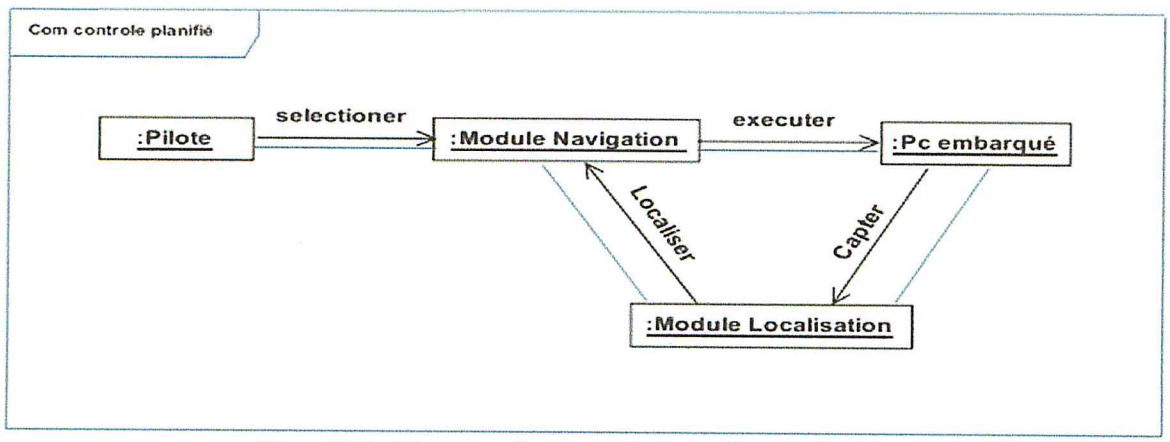
A la fin de l'exécution le système demande au pilote de nouvel requête à accomplir est reste en attente pour de nouvelle mission.

III.5.3 Cas d'utilisation «Contrôle planifié»

Le but de ce mode est de permettre au pilote de planifier au robot une trajectoire à parcourir en utilisant le module de localisation de robot dans son environnement.
Afin d'effectuer ce mode le pilote choisi le mode planification et insère les vitesses des roues nécessaire à la trajectoire que le robot va parcourir, et les vitesses des axes du bras-manipulateur à exécuter lors de l'arrivé au but.

III.5.3.i Diagramme de communication « Contrôle planifié» :

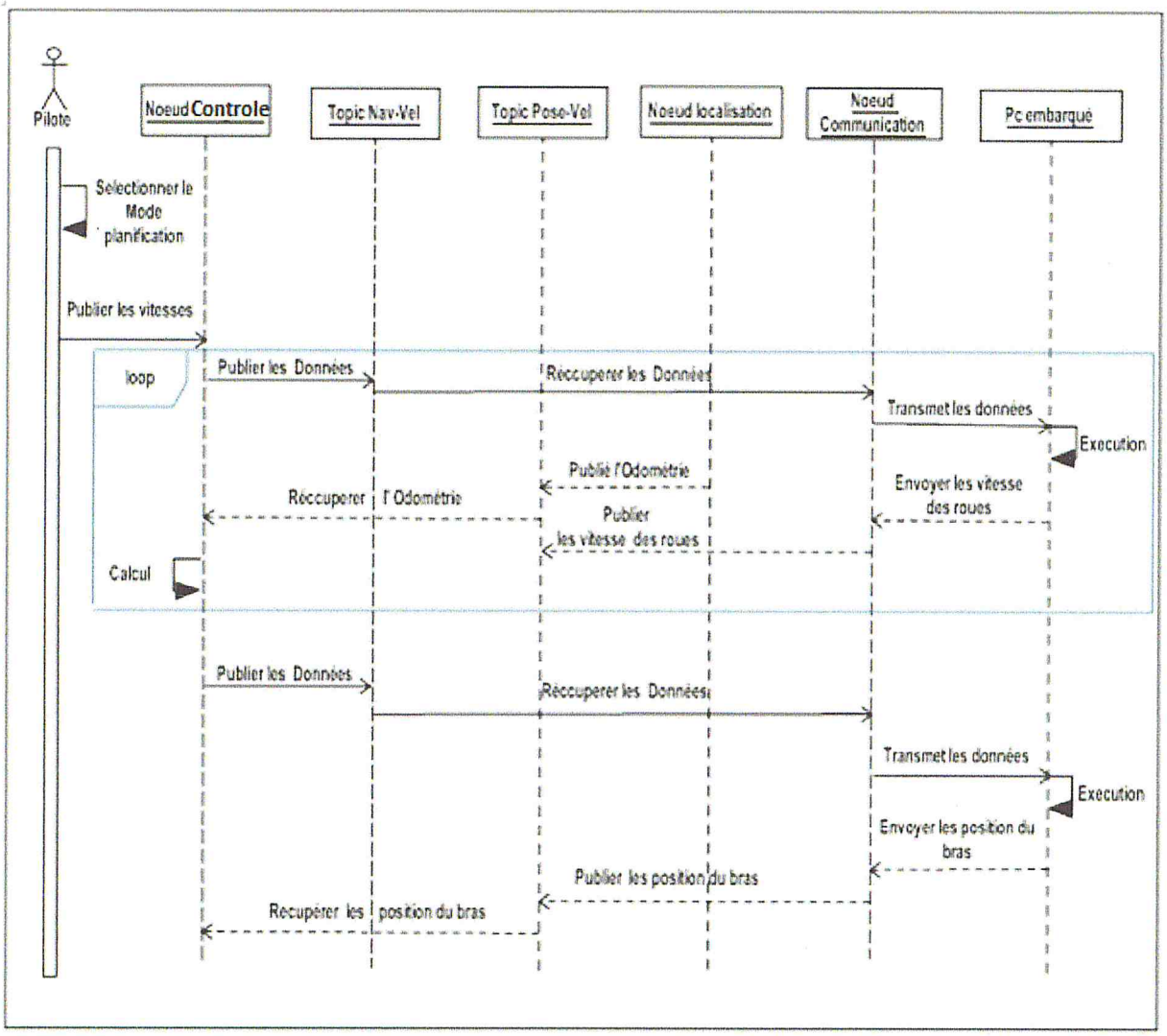
Ce diagramme résume les principaux modules impliqués dans le mode planifié. Les trois module : contrôle, pc-embarqué et localisation ne s'arrêtent pas de communiquer en boucle selon des interactions spécifiques afin d'assurer la circulation des données essentiel.



FigureIII.7-- Diagramme de communication « Contrôle Planifié »

III.5.3.ii Diagramme de séquence « Contrôle planifié» :

C e diagramme nous permet montrer les interactions entre le pilote et le système dans le cadre du scénario control planifié



FigureIII.8 --Diagramme de séquence « Contrôle planifié »

Scénario « Contrôle planifié»

En premier temps Le pilote sélectionne le mode de contrôle planifié et saisit les vitesses des roues pour la base mobile, ensuite les informations saisie sont envoyées au nœud control. Ce dernier effectue le traitement des données reçues et génère les mouvements nécessaires pour cette requête. Après que ces données soient publier dans le topic nav-vel, celui-ci s'abonne par le nœud communication qui a son tour récupère ces données de ce topic et les transmet au pc embarquée via socket tcp/ip.

La partie embarqué située dans le pc embarqué réceptionne les commandes et les affecte aux actionneurs pour l'exécution.Quand le robot commence de mouvoir il renvoie les vitesses des roues via réseau tcp/ip au nœud communication. Ce dernier publie ces données dans le topic pose-vel.

Un nouveau nœud est introduit dans ce mode, c'est le nœud Localisation. Ce nœud publie la localisation du robot (X,Y ,ɵ) dans le topic pose-vel. Celui-ci est abonné au nœud control qui, â son tour récupère les données contenus dans ce topic, recalcule et vérifie le bon déroulement de la mission et régénère les commandes nécessaires.

A l'arrivé au but la base mobile s'arrête et le bras manipulateur commence sa mission en exécutant la position introduit par le pilote.

III.5.4Cas d'utilisation «Contrôle autonome»

Le but de ce mode est de permettre au pilote de donner au robot un but à atteindre dans son environnement sans lui montrer la trajectoire en utilisant le module de localisation du robot et le module capteur LMS pour l'énvironnement d'obstacle.

Afin d'effectuer ce control le pilote choisi le mode autonome en insérant la position cible de la base mobile dans l'enivrement, et les positions du bras manipulateur.

III.5.4.i Diagramme de communication « Contrôle autonome»

Ce diagramme résume les principaux modules impliqués dans le mode planifié. Les trois module : contrôle, pc-embarqué et localisation ne s'arrêtent pas de communiquer afin d'assurer la circulation des données essentielles.

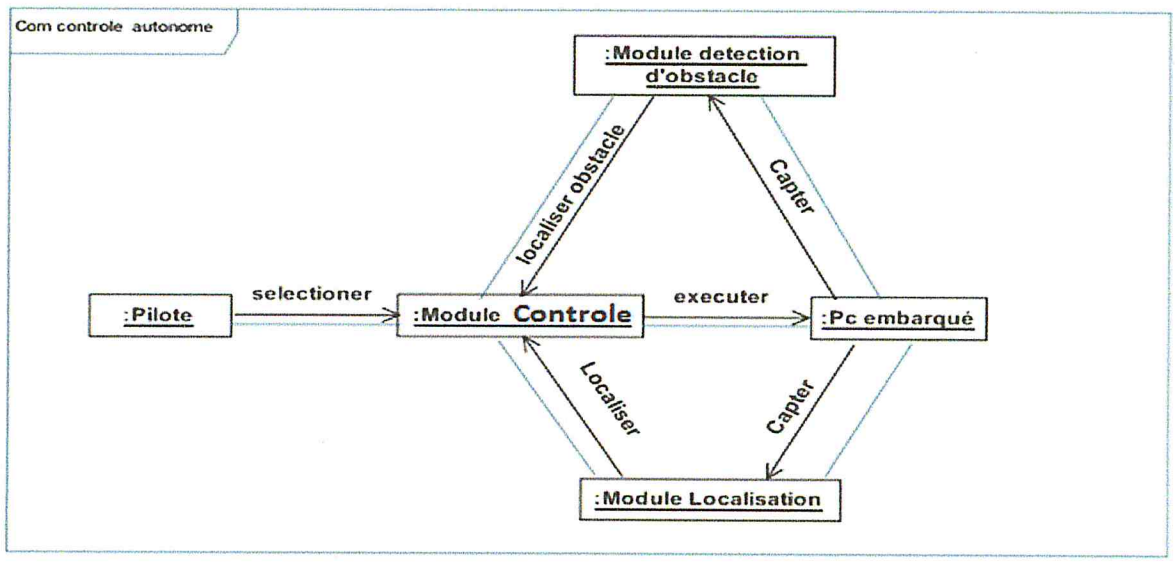


Figure III.9- Diagramme de communication « Contrôle hybride »

III.5.4.ii Diagramme de séquence « Contrôle autonome» :

Ce diagramme permet de visualiser l'enchainement des action dans le temps entre les différent objet dans le temps lors de l'exécution du mode autonome

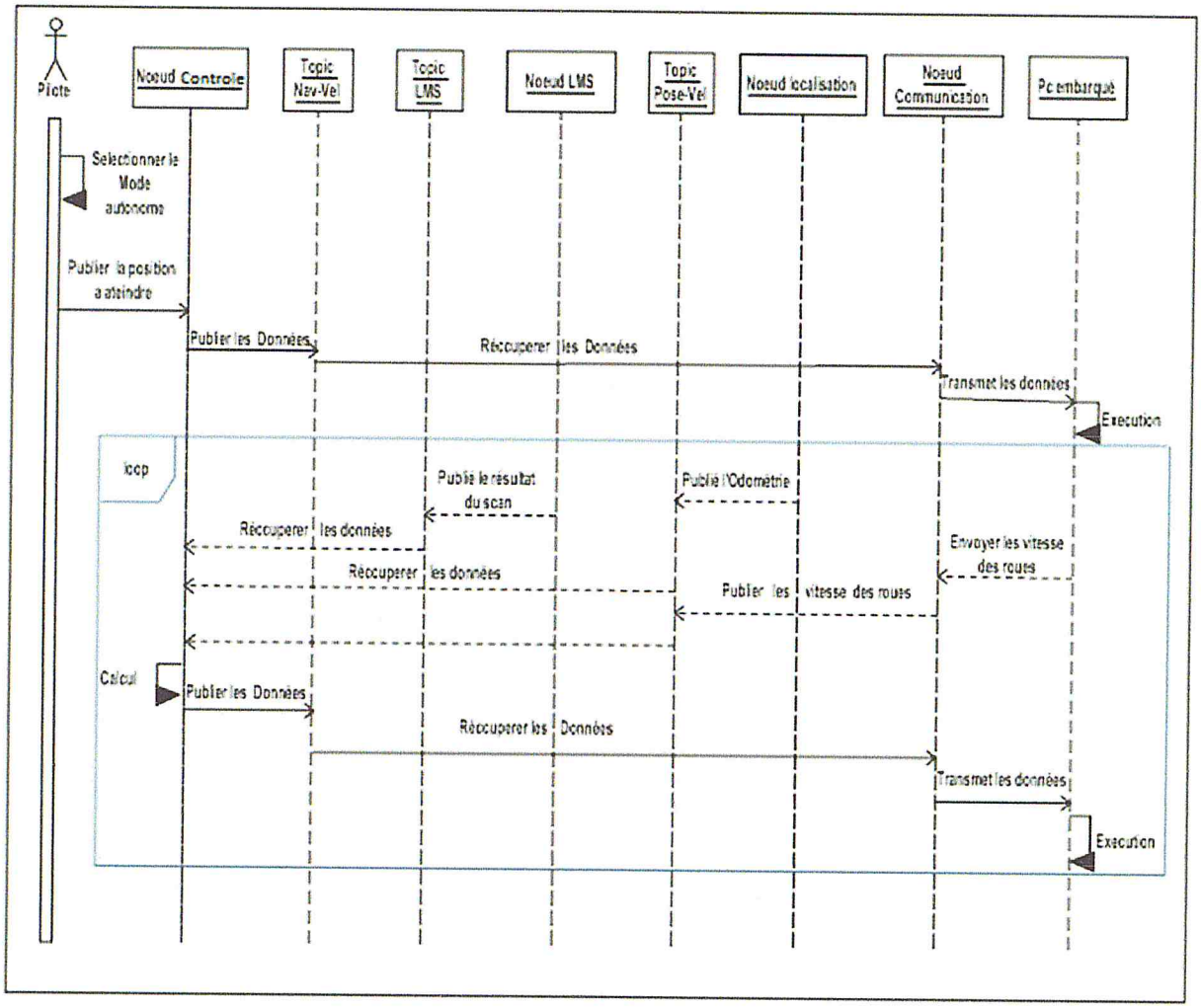


Figure III.10-- Diagramme de séquence « Contrôle autonome»

Scénario « Contrôle autonome»

Dans un premier temps, le pilote sélectionne le mode de contrôle autonome et saisie la position cible pour la base mobile et les positions des axes du bras manipulateur. Ensuite les informations saisies sont envoyées au nœud control. Ce dernier traite les données reçues par le pilote et génère les mouvements nécessaire pour cette requête,

La partie embarquée située dans le pc embarqué réceptionne les commandes et les affecte aux actionneurs pour l'exécution.

Quand le robot commence de mouvoir il envoie les vitesses des roues via réseau tcp/ip au nœud communication. Ce dernier publie ces données dans le topic pose-vel.

Ensuite le nœud Localisation publie la localisation du robot (X,Y ,theta) dans topic-vel.

Un nouveau nœud est introduit dans ce mode, c'est le nœud capteur laser LMS qui récupère les signaux à partir du laser du robot Robuter ULM et publie ces données dans le topic Topic-LMS. Celui-ci est abonné par le nœud control qui à son tour récupère les données contenue dans ce topic.

Le nœud control recalcule et vérifie le bonne déroulement de la mission et régénère les commande nécessaire et un nouveau cycle débutera jusqu'à l'arrivé au but.

A l'arrivés au but la base mobile s'arrête et le bras manipulateur commence sa mission en exécutant la position introduit par le pilote.

III.5 Conclusion

Dans ce chapitre on a présenté une architecture de control d'un robot manipulateur mobile appliqué sur le Robuter ULM. Cette nouvelle architecture et différente de l'ancienne architecture matérielle et logicielle du robot. Les modes de contrôle du robot à travers la nouvelle architecture sont modélisées en UML en utilisant les digrammes de classes, de communication et de séquence. Le chapitre suivant portera sur l'implémentation et les tests de l'architecture.

Chapitre IV

Implémentation et test

CHAPITRE IV : IMPLEMENTATION ET TEST

IV.1 Introduction

Le but de projet est de réaliser une architecture de génération de mouvement pour un robot manipulateur mobile. Le robot sur lequel a été réalisée l'implémentation de ce projet est le Robuter/ULM. Ce dernier est un robot tournant sous un OS linux et dont la structure matérielle est relativement modeste, mais suffisante pour la réalisation des diffèrent requête envoyées.

En sachant que la configuration matérielle ne permet pas l'installation de ROS directement sur le PC embarqué du Roboter/ULM, il fallait trouver une solution pour pouvoir utiliser ROS sur cette plateforme, et la solution qui a été adoptée, était une solution qui nécessitait l'utilisation d'un pc externe au robot sur lequel est installé ROS ainsi qu'établir une connexion client/serveur à distance entre le PC ROS et le PC Embarqué avec le protocole TCP/IP.

Dans ce chapitre nous exposons les différents programmes mis en œuvre lors de ce projet et leur implémentation sur le robot afin d'atteindre notre objectif qui est l'exploitation des fonctionnalités de ROS à distance sur Robuter/ULM.

IV.2 Le protocole de communication

Nous avons connecté le pc distant dans le réseau local du laboratoire de la division robotique du CDTA par une liaison wifi, dont tous les robots du laboratoire sont connectés et éventuellement Robuter/ULM est connecté dans ce réseau. la communication entre ce dernier et le PC ROS se fait via un système de socket et un protocole tcp/ip . Dans les domaines de la télécommunication et des réseaux informatiques. Nous avons utilisé le protocole de communication le plus répondu et le plus utilisé dans le monde à savoir le protocole tcp/ip.

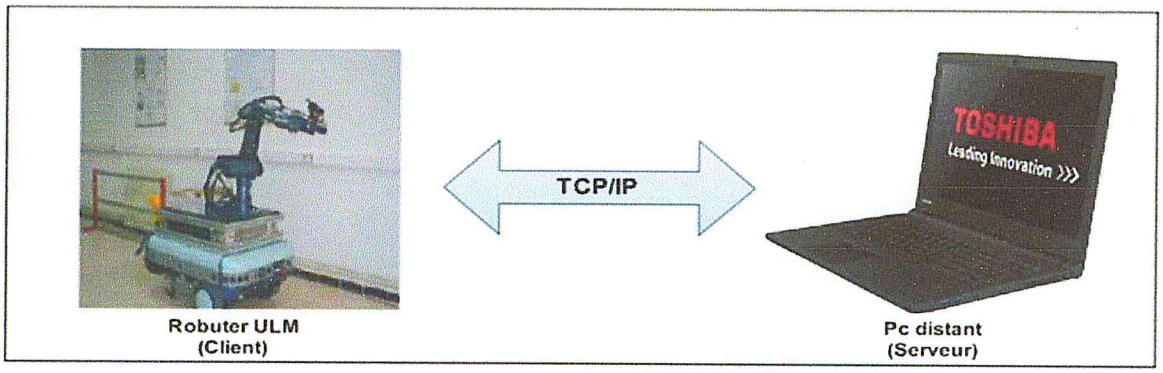


Figure IV.1—Principe de communication Robot/Pc distant

IV.2.1 Le modèle client /serveur :

TCP est peer-to-peer (au sens particulier à particulier), c'est-à-dire d'un ordinateur à un autre sans qu'il y est un maitre/esclave, cependant, pour permette la communication entre deux ordinateur (qui nous servira à ajouter des application) on utilise généralement un modele appeler client/serveur (figure IV.2).

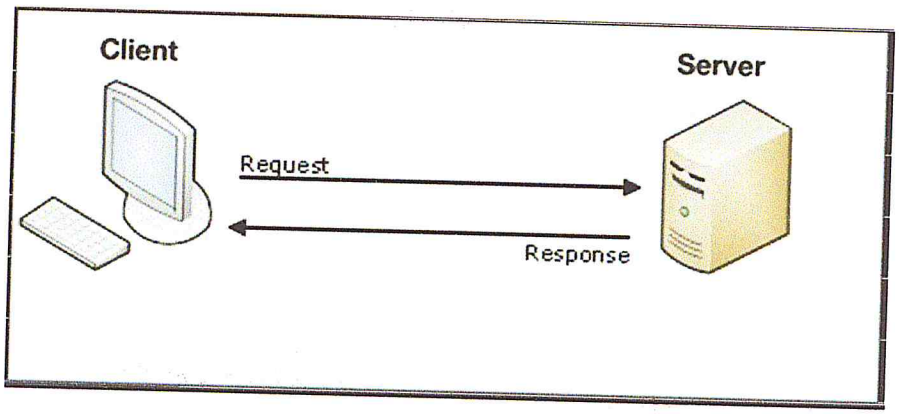


Figure IV.2—Modèle d'application client/serveur[36]

- **Le serveur :** C'est une application (programme écrie en C++ dans notre cas) qui offre un ou plusieurs services au client à la réception d'une requête, déclenche les processus associé au service, puis envoie la réponse vers le client.

  Dans notre cas le serveur se déclenche lorsque le pilote active l'un des trois modes de control de robot (chapitre III), envoie les commandes au client et réceptionne les données des capteurs proprioceptifs retourné par le client du Robot ULM.

- **Le client :** C'est une application qui envoie une requête au serveur dans l'attente d'une repense (au besoin de ce service), dans notre cas le traitement sont effectués majoritairement sur le serveur, le travail du client se focalise sur la réception des commande du serveur, les transmettre au actionneur pour finaliser l'action demander ainsi que renvoyé les données des capteurs proprioceptif au poste serveur.

IV.3 Langage utilisé :

On a utilisé Pour notre architecture :

IV.3.1 Langage C++ :

Le C++ est un langage de programmation permettant la programmation sous de multiples paradigmes comme la programmation procédurale, la programmation orientée objet et la programmation générique.

C++ est actuellement le 3e langage le plus utilisé au monde. Le langage C++ n'appartient à personne et par conséquent n'importe qui peut l'utiliser sans besoin d'une autorisation ou obligation de payer pour avoir le droit d'utilisation.

IV.3.2 Langage C :

Le C est un langage de programmation impératif, modulaire et structuré, conçu pour la programmation au niveau du système d'exploitation. Le C est un langage aussi dit de bas niveau, ce qui veut dire qu'il est très proche du fonctionnement de l'ordinateur.

Il se distingue par le fait que tous ses concepts ont une traduction simple en langage machine, qui ne nécessite ni bibliothèque ni machine virtuelle pour assister l'exécution.

IV.4 Les sockets

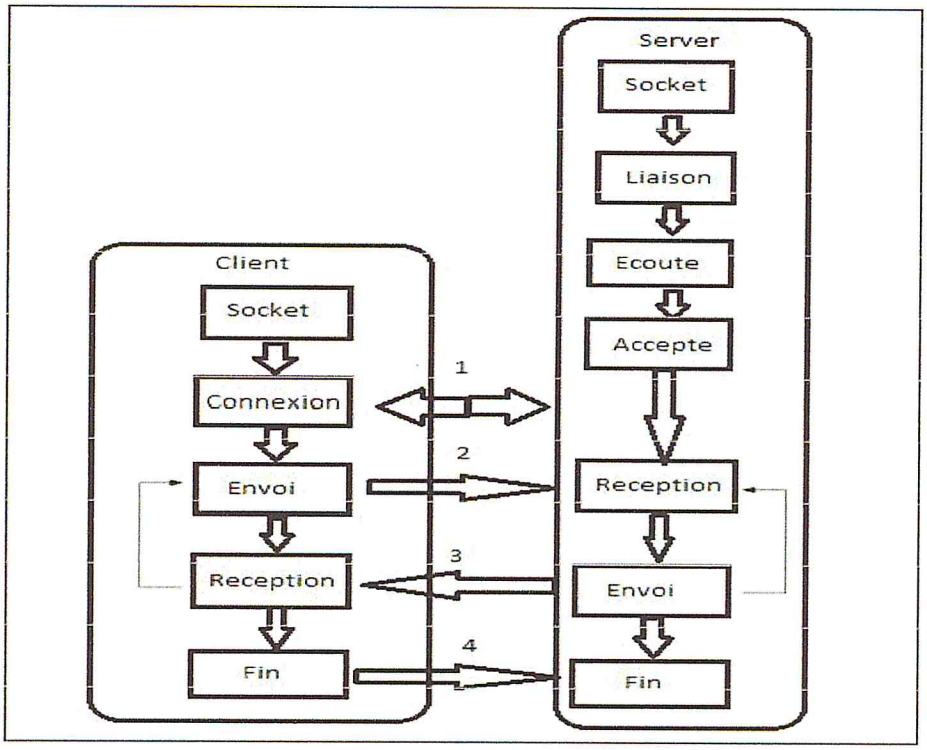
Comme nous travaillons sous une architecture LINUX et que nous employons le protocole de communication TCP/IP, on a intégré des sockets à nos programmes client-serveur.

Le terme socket est utilisé pour définir le récipient ou le convoyeur de la donnée envoyée.

Pour comprendre le fonctionnement des sockets, on donnera pour exemple les minerais qui doivent être transporté de la mine jusqu'au broyeur dans un chariot, de même pour les socket qui transportent les messages (données ou informations) sur le réseau, elle doivent être créer par une commande spécifique « sock() », la fonction « bind() » spécifie l'adresse correspondent à chaque socket afin d'envoyer et recevoir des messages.

La fonction « listen() »permet de mettre le programme en écoute des requête de communication, des qu'une demande de communication ce présente notre fonction « accept() » assure la connexion.

Quand toutes ces étapes seront franchies (sans erreurs), les fonctions « send » et « recev » (selon le besoin de l'utilisateur) s'occupent de la convention (l'échange de message) qui sera finalisé par la commande « close » (voire figure IV.3).



FigureIV.3 -- Fonctionnement des sockets[24]

IV.5 Les threads

Un thread ou fil ou tache, est similaire à un processus car tous deux représentent l'exécution d'un ensemble d'instruction du langage machine d'un processus. Du point de vue de l'utilisateur, ces exécutions semblent se dérouler en parallèle.

L'utilisation de plusieurs threads permet de paralléliser leur exécution ce qui permet de l'effectuer plus rapidement. Les threads permet aussi le partage de ressources entre eux ce qui entraine une communication plus efficace.

IV.5.1 Les thread du PC ROS

Dans cette section en présente les différant thread qui forme les nœuds de côté serveur.

IV.5.1.i Les threads du nœud communication (partie serveur)

Ce nœud comporte 4 thread qui s'exécutent en parallèle et échange les données entre aux.

-Le thread subscriber () : Ce thread (implémenter sur le serveur) permet au programme serveur de s'abonner au topic NAV-VEL et récupérer les données publiées par le nœud Navigation en temps réel (voir Figure IV4).

```
ros::Subscriber sub ;

void velocityCallback(const  std_msgs::Float64MultiArray::ConstPtr& vel_msg)
 {
  for(j = 0; j < 15; j++){
 arr[j] = vel_msg->data[j];
 }

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

    ros::init(argc, argv, "srv");
    ros::NodeHandle n;
    sub = n.subscribe("Nav-Vel", 1000, velocityCallback);
    ros::spin();
         ...
}
```

FigureIV.4-- structure général du Thread subscriber

-Le thread talker () : C'est l'un des plus important puisqu'il il permet de communiquer avec la partie client (pc embarqué) afin d'envoyé les diffèrent information relative à la navigation et qui sont récupérer par le thread subscriber (voir Figure IV5).

```
static void* talker (void* unused)
{   ...
server = accept(client,(struct sockaddr *)&server_addr,&size);
 while (server > 0)

    {
      sprintf(neww,"%9lf",arr[0]);
      strcpy(buffer,neww);
      send(server, buffer, bufsize, 0)
     }
...
}
int main(int argc, char **argv)
{
  pthread_t thread_id;
       pthread_create (&thread_id, NULL, &talker, NULL);
   ...
pthread_join(thread_id,NULL);
....
}
```

FigureIV.5-- structure général du Thread talker

-Le thread listner () : Ce thread joue aussi un rôle très important car il communique avec la partie client et réceptionne les données relatives aux capteurs proprioceptifs du Robuter/ULM (voir Figure IV.6).

```
static void *listner(void* unused)
{
...
   server = accept(sock,(struct sockaddr *)&server_addr,&size);
...
 while (server > 0)
    {
        while(cpt<14){
         rc =recv(server,buffer,bufsize,0);
      rcv[cpt]=atof(buffer);
        printf("\nrcv= %f " , rcv[cpt]);
                cpt++; }
...
int main(int argc, char **argv)
{
 pthread_t tlistner;
  pthread_create(&tlistner, NULL, &listner, NULL);
   ...
pthread_join(tlistner,NULL);
....
}
```

Figure IV.6-- structure général du Thread listner

-Le thread Publisher () : Ce thread publie les données réceptionnées par le thread listner dans le topic Pose-Vel (voir Figure IV.7).

```
static void *publisher( void* unused)
{
 std_msgs::Float64MultiArray vel_msg1;
...
ros::Rate loop_rate(10);
   do{

    velocity_publisher.publish(vel_msg1);
    ros::spinOnce();
  loop_rate.sleep();
  }while(1);
return NULL ;
}
int main(int argc, char **argv)
{
 pthread_t pub_id;
pthread_create(&pub_id, NULL, &publisher, NULL);
   ...
pthread_join(pub_id,NULL);
....
}
```

Figure IV.7-- structure général du Thread publisher

IV.5.1.ii Les threads du nœud navigation

Le nœud navigation contient trois threads :

- Thread Publisher.
- Thread subscriber position.
- Thread subscriber laser.

-Thread Publisher : ce thread publie les données relatives à la base mobile et au bras manipulateur dans le topic NAV-VEL

-Thread Subscriber position : C'est le thread qui récupère les données relatives à la localisation du robot ULM

-Thread subscriber laser : C'est le thread qui récupère les données du capteur laser à partir du topic LMS.

IV.5.1.iii Le threads du nœud localisation

-Thread Publisher loc : Le nœud localisation contient un thread qui a pour rôle de publier les données de localisation du robot ULM dans le topic Pose-Vel.

V.5.1.iv Le threads du nœud capteur LMS :

-Thread Publisher : Ce nœud publie les informations relatives aux scanners du capteur LMS, ces données sont publier dans le topic LMS en temps réels (voir Figure IV.8).

```
void publish_scan(diagnostic_updater::DiagnosedPublisher<sensor_msgs::LaserScan> *pub, uint32_t *range_values,
                  uint32_t n_range_values, uint32_t *intensity_values,
                  uint32_t n_intensity_values, double scale, ros::Time start,
                  double scan_time, bool inverted, float angle_min,
                  float angle_max, std::string frame_id)
...
int main(int argc, char **argv)
{
.....
publish_scan(&scan_pub, range_values, n_range_values, intensity_values,
                   n_intensity_values, scale, start, scan_time, inverted,
                   angle_min, angle_max, frame_id);
                        ros::spinOnce();
....
}
```

FigureIV.8 -- structure général du Thread publish_scan

IV.5.2 Les threads du PC EMBARQUE

-Le thread listner client () : Ce thread joue un rôle très important car il communique avec la partie serveur et réceptionne les données relatives au roues de la plateforme mobile et le bras manipulateur Robuter/ULM

-Le thread talker () : Ce thread permet de communiquer avec la partie serveur (pc ROS) afin d'envoyé les différent information relative au différents capteur.

IV.6 Interface Homme/Robot

Interface terminal : Il est parfois plus simple de taper une commande que d'effectuer des manipulations demandant beaucoup de clics de souris dans une interface graphique. C'est aussi un moyen plus simple pour expliquer comment faire quelque chose à quelqu'un, puisque il suffit d'indiquer la commande et non la suite de clics à effectuer sur l'interface graphique.

Cependant, même si le terminal peut être beaucoup plus efficace qu'une interface graphique sous les doigts d'un utilisateur avancé, il est moins abordable que les interfaces graphiques.Il est probable qu'aucune des deux méthodes (commandes ou interface graphique) ne remplacera complètement l'autre car elles se complètent plus qu'elles ne rivalisent.

L'utilisation est assez simple, tapez une commande (ou copiez-collez la) et faites *Entrée* (clavier) pour l'exécuter.Les raccourcis pour le copier-coller ne sont pas Crtl+C ↔ Ctrl+V par défaut, mais Maj+Ctrl+C et Maj+Ctrl+V. En effet, dans un terminal le raccourci Ctrl+C annule la commande en cours. Cependant il est possible de modifier les raccourcis du Terminal (voir Figure IV.9)

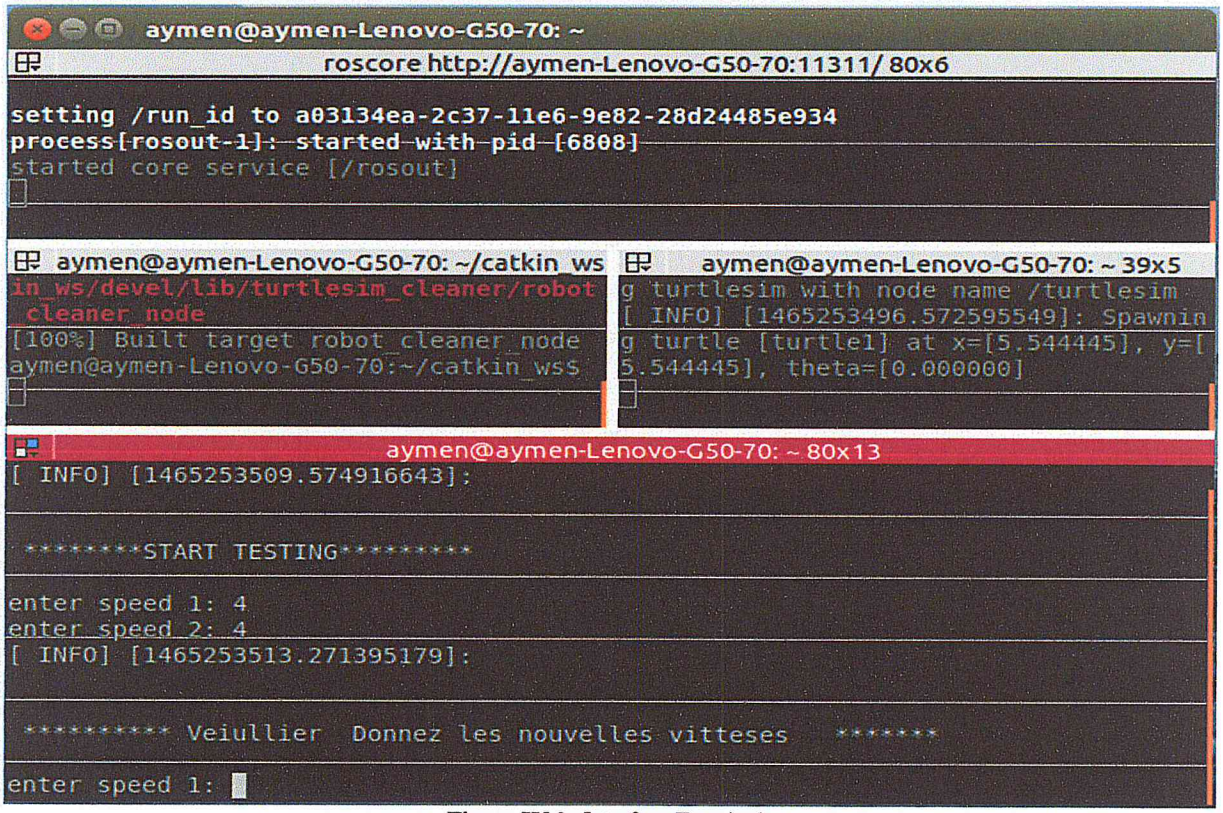


Figure IV.9--Interface Terminal

Première commande : lancement obligatoire du nœud master par la commande roscore (voire Figure IV.10).

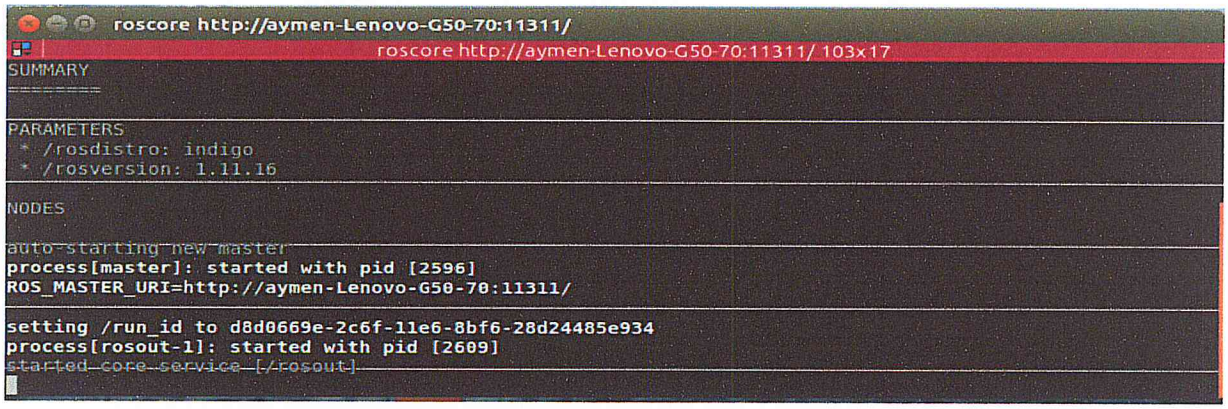


Figure IV.10—Lancement du nœud master

Deuxième commande : Changement de répertoire par la commande (cd) suivi par le nom du work_space ,après faire le Make de ce dernier pour la génération des exécutables et des dépendances des nœuds par la commande catkin_make (voir Figure IV.11) .

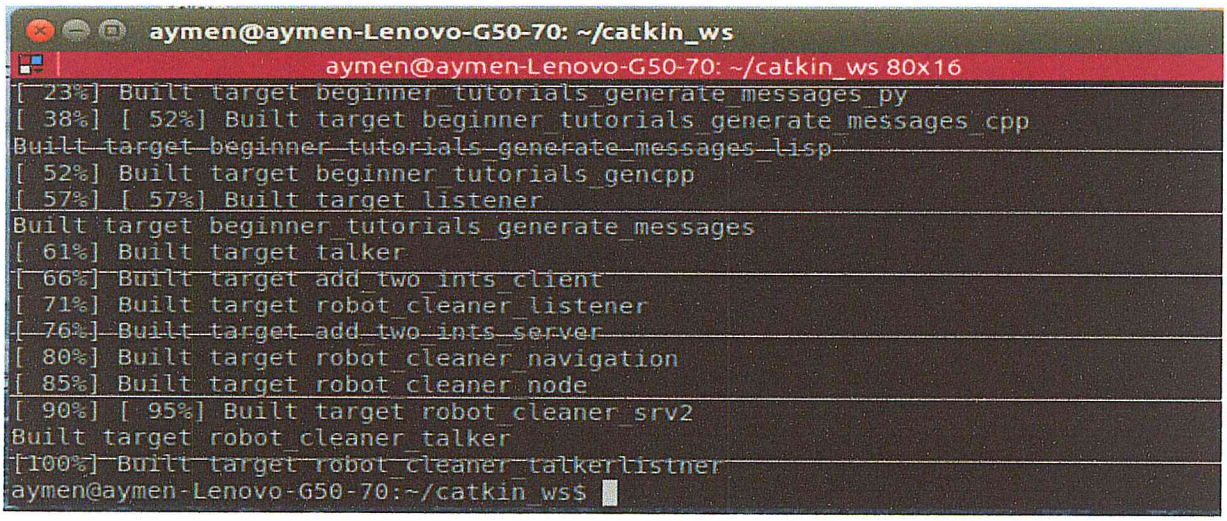


Figure V.11—Make du workspace

Troisième commande : lancement des nœuds (exécution d'un ou plusieurs) par la commande rosrun en précisant le nom du package ou se trouve le nœud à lancer et le nom de l'exécutable(voir Figure IV.12).

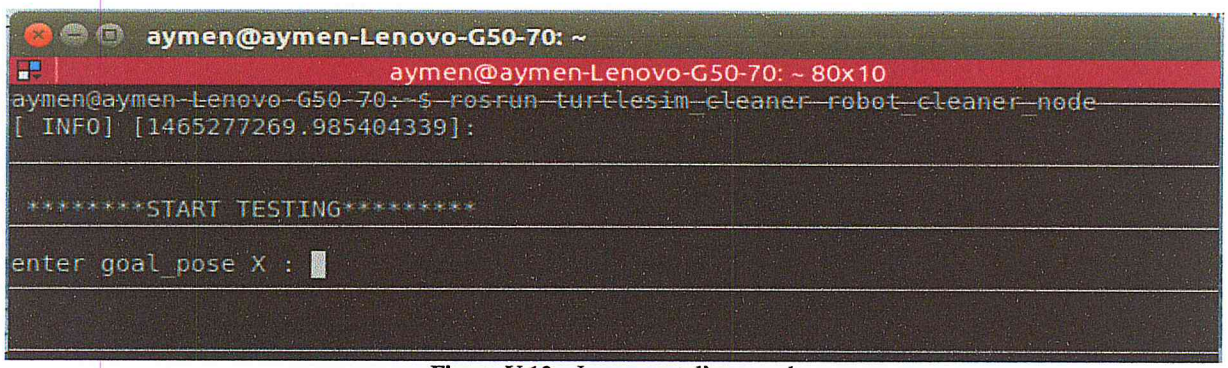


Figure V.12—Lancement d'un nœud

IV.7 Tests et résultats

Afin de pouvoir tester l'architecture proposée en a fait une simulation des trois modes et des tests sur le robot.

On a deux partie majeurs à tester, la première partie concerne la vérification de l'établissement de connexion entre la partie implémentée sur le robot et celle implémentée sur le pc distant.

La deuxième partie et tester les modes de contrôle parle simulateur turtlesim, et tester réellement sur robot Robuter/ULM le mode planifié.

IV.7.1 Communication Robot/pc distant

Dans cette figure on peut voir l'arrivées des données proprioceptif :les vitesses des roues, vitesses et les positions des axes du bras manipulateur, la localisation(Odométrie) au nœud navigation(voir Figure IV.13).

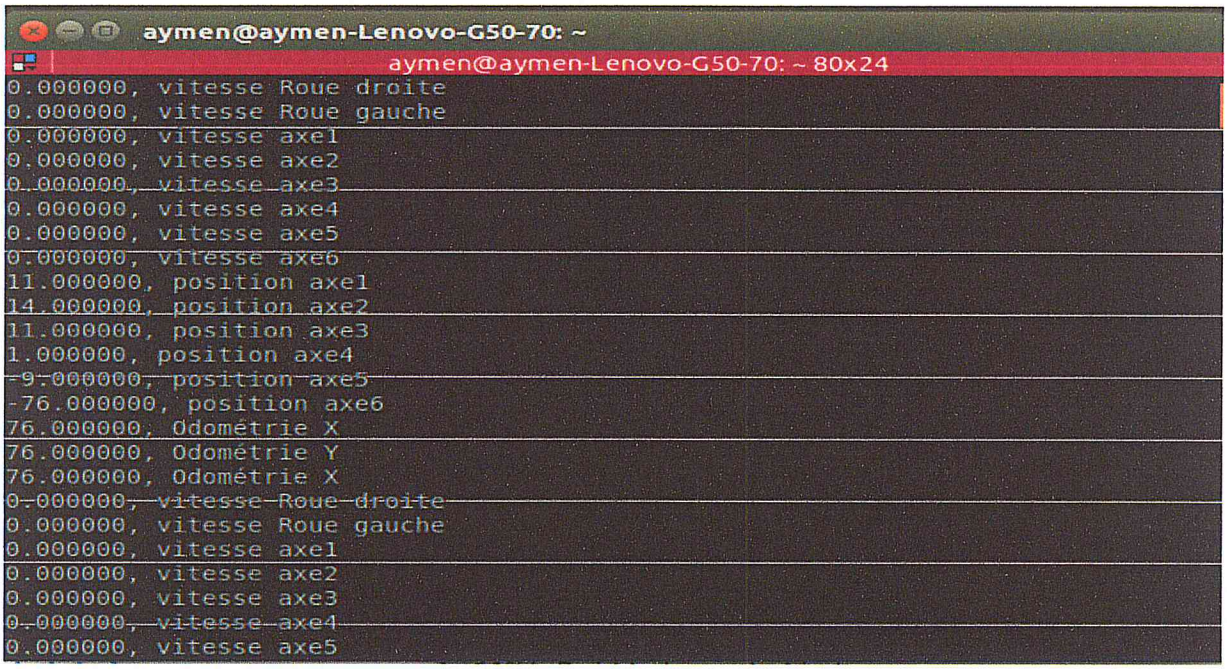


Figure IV.13--Réception des données du capteur proprioceptif par le nœud navigation

Descriptif de la figure :

Réception des données concernant la base mobile et le bras manipulateur à partir du serveur (pc_ROS) pour l'exécution(voir Figure IV.14).

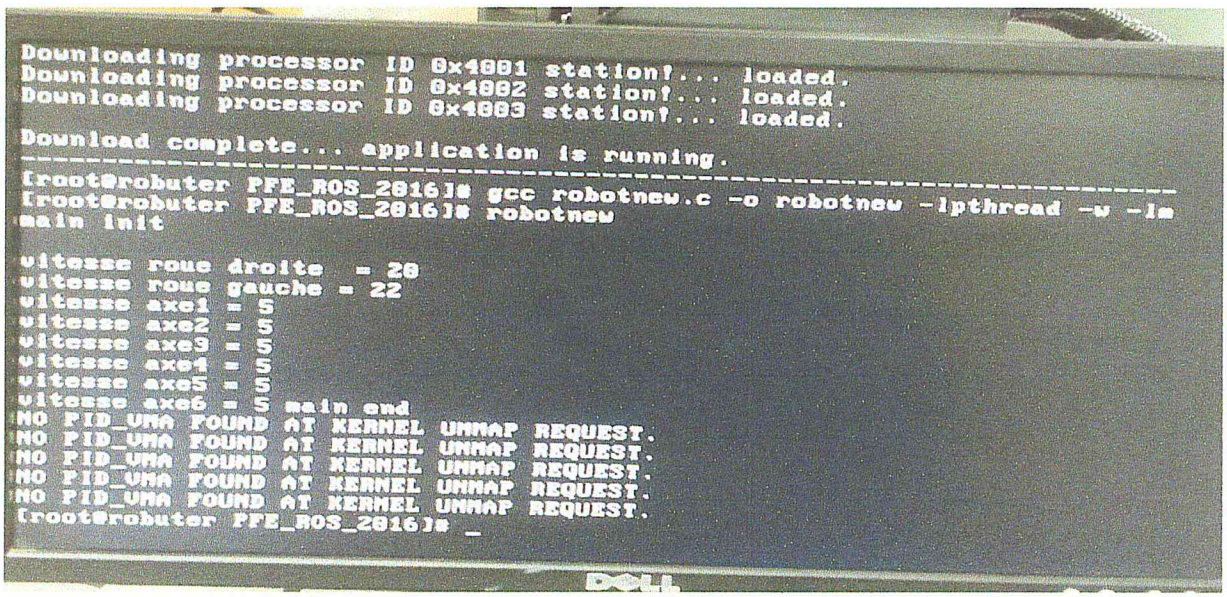


Figure IV.14--réception des données de mouvement par le pc embarqué.

Descriptif de la figure :

Lors de l'activation du nœud LMS le scanner laser (voir figure IV.15) effectue un balayage de l'environnement et retourné les données captés, le nœud LMS publie ces données dans le topic scan LMS (voir figure IV.16).

On peut visualiser le résultat du scan par l'outil fournie par ROS appeler RVIZ, la figure LMS (voir figureIV.17) nous permet de visualiser la map générer RVIZ. ON voir les obstacles situé devant le robot robuter/ULM.

FigureIV.15--Scanner laser LMS

```
nassim@nassim-Satellite-C850-B907: ~ 81x32
intensities: []
---
header:
  seq: 197
  stamp:
    secs: 1465210743
    nsecs: 969827280
  frame_id: laser
angle_min: -1.57079637051
angle_max: 1.57079637051
angle_increment: 0.0174532923847
time_increment: 3.70370362361e-05
scan_time: 0.0133333336562
range_min: 0.0
range_max: 8.10000038147
ranges: [1.5430001020431519, 1.5420000553131104, 1.5420000553131104, 1.542000055
3131104, 1.5380001068115234, 1.5500000715255737, 1.5580000877380037, 1.5570000410
079956, 1.568000078201294, 1.5760000944137573, 1.5760000944137573, 1.58700013160
70557, 1.5830000638961792, 1.5980000495910645, 1.6880000829696655, 1.98400008678
43628, 2.005000114440918, 2.01200008392334, 2.0210001468658447, 2.03100013732910
16, 2.0400002002716064, 2.06000018119812, 2.068000078201294, 2.0870001316070557,
 2.0940001010894775, 2.1050000190734863, 1.6340000629425049, 1.6200001239776611,
 1.6100001335144043, 1.6210000514984131, 1.6720000505447388, 2.254000186920166, 2
.2870001792907715, 1.5310000181198120, 1.5220000743865967, 1.5050001144409180, 1.4
930000305175781, 1.5080000162124634, 2.4710001945495605, 2.4910001754760742, 2.40
70000648498535, 2.3890001773834229, 2.4300000667572021, 2.4560000896453857, 2.5010
00165939331, 2.5370001792907715, 2.5830001831054688, 2.6260001659393311, 2.670000
0762939453, 2.7260000705718994, 2.7890000343322754, 2.8390002250671387, 2.891000
0324249268, 3.1870000362396240, 3.2720000743865967, 3.3590002059936523, 3.4380002
02178955, 3.5250000953674316, 3.6220002174377441, 3.7270002365112305, 3.832000255
584717, 3.9530000686645508, 4.0760002136230469, 4.2120003700256348, 4.3630003929138
18, 4.5190000534057617, 4.6870002746582031, 4.8770003318786621, 5.0780000686645508,
```

Figure IV.16 --les données publier dans le topic Scan LMS

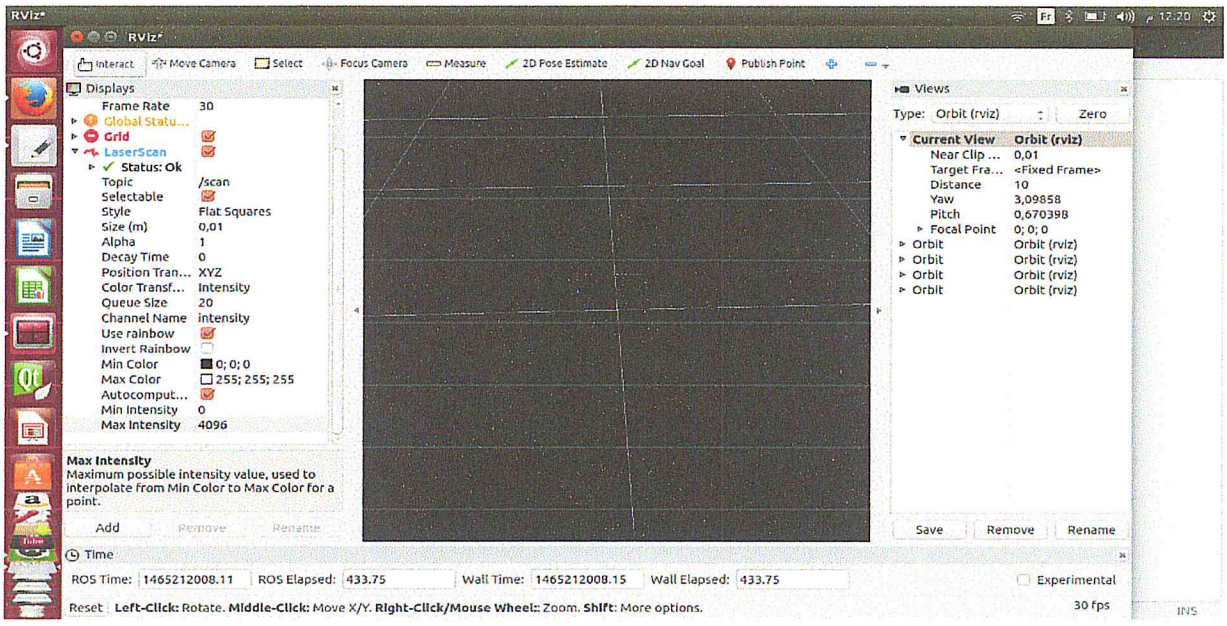


Figure IV.17 --interface RVIZ

IV.7.2 Simulation turtlesim :

On a simulé nos trois modes de control de robot par le simulateur turtlesim.

- **Mode manuel**

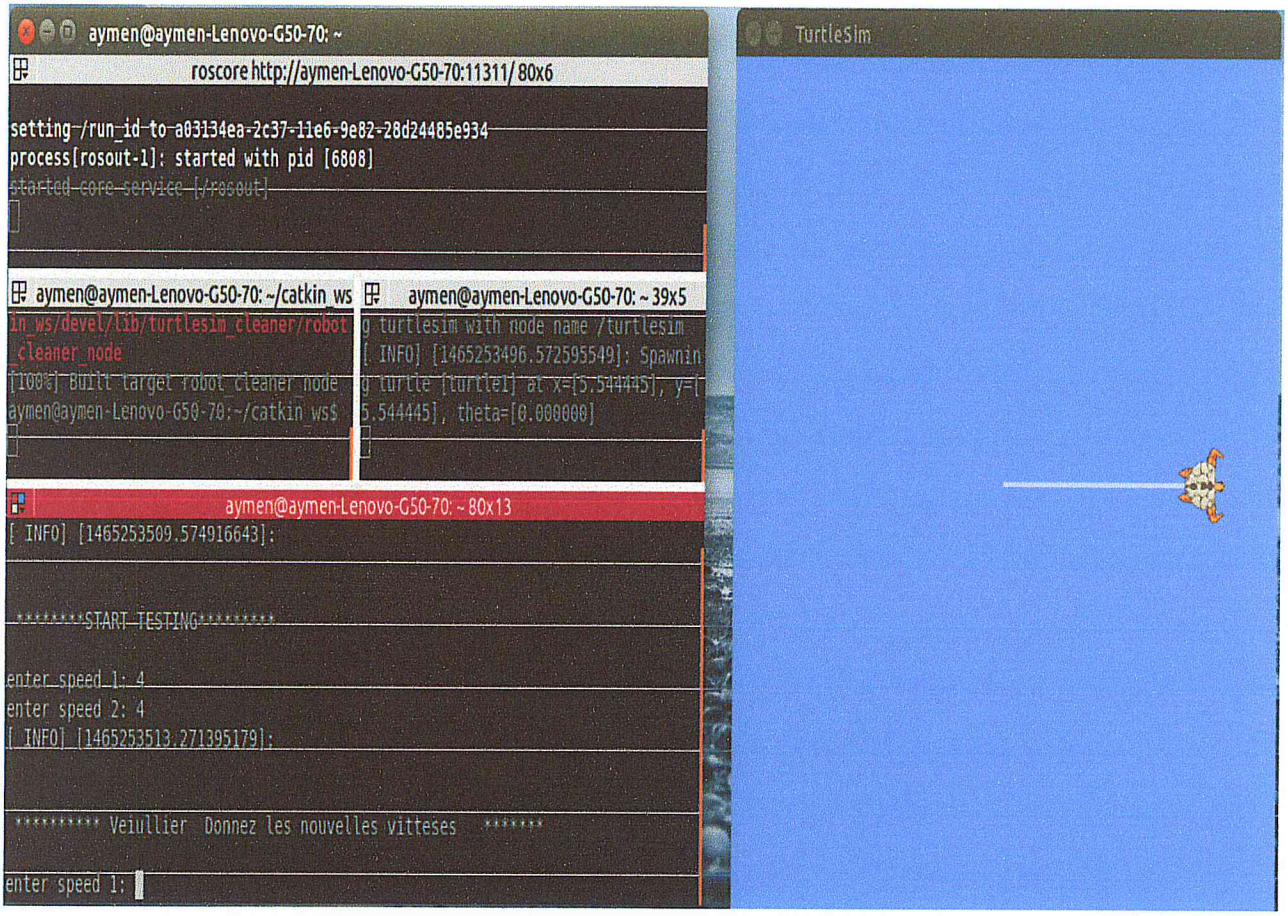


Figure IV.18-- Simulation du mode manuel avec turtlesim.

Le Pilote lance d'abord le nœud master (roscore), puis le make du workspace ensuite lance les deux nœuds turtlesim et navigation.

Après que le pilote saisie les vitesses le turtle exécute le mouvement et redonnes la main au pilote pour une nouvelles requête (voir la figure IV.18).

- **Mode planification**

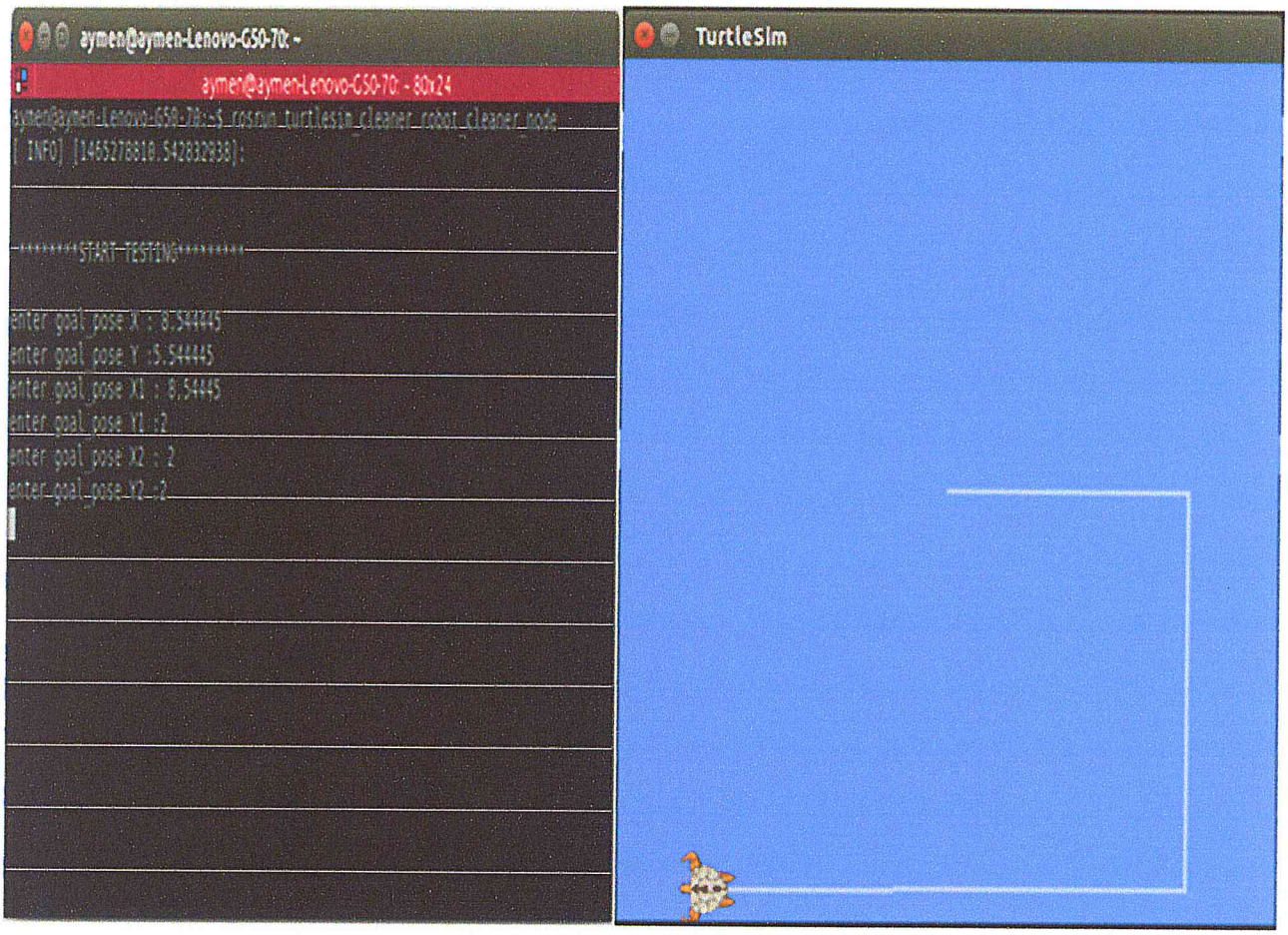


Figure IV.19 -- Simulation du mode planifié avec turtlesim

Le Pilote lance d'abord le nœud master (roscore), puis le make du workspace ensuite lance les deux nœuds turtlesim et navigation. Après que le pilote saisie une suite de vitesses pour que le robot parcourt un chemin voulue, le turtle exécute le mouvement et s'arrête après l'exécution des dernière vitesses (voir la figure IV.19).

- **Mode autonome :**

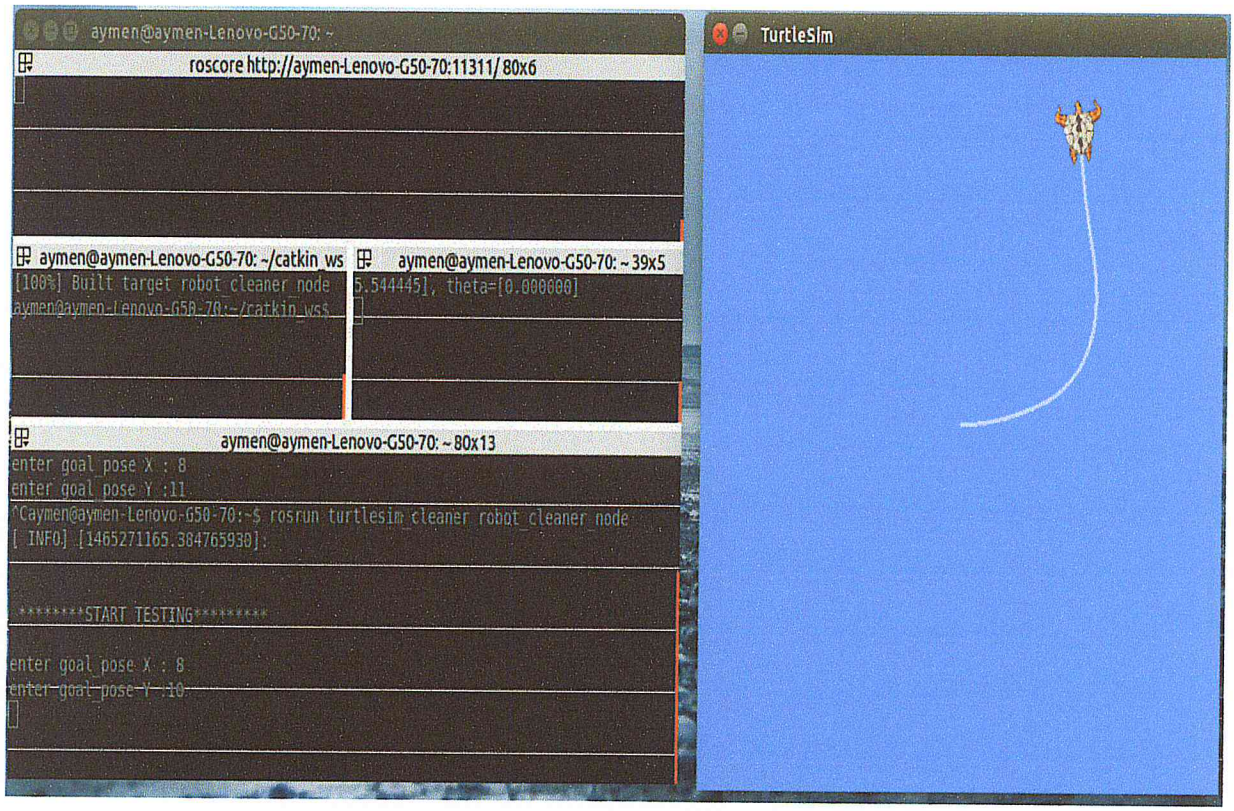


Figure IV.20-- Simulation du mode autonome avec turtlesim.

Le Pilote lance d'abord le nœud master (roscore), puis le make du workspace ensuite lance les deux nœuds turtlesim et navigation. Après que le pilote saisie la position (X,Y) cible dans l'environnement, le turtle exécute le mouvement et s'arrête des qu'il attend le but (voir la figure IV.20).

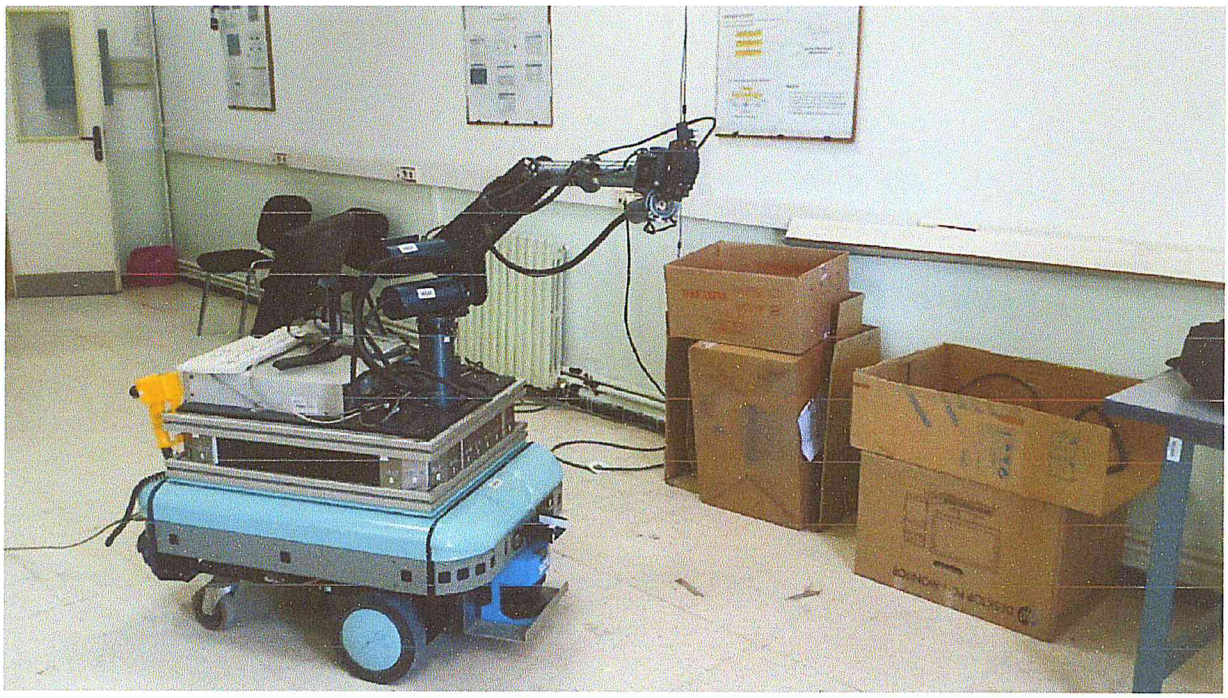
IV.7.3 Test sur Robuter ULM

Descriptif des figures IV.21 ET IV.22 :

Dans ce test on effectuera la communication entre le PC ROS et le PC embarqué, en envoyant des vitesses au roues et au axes du bras manipulateur, les données ont était bien réceptionné par le robot et ont était affecté au actionneur.

Le robot a démarré d'une position initiale (voir figure IV.21) pour la base mobile et pour le bras manipulateur, après l'exécution le robot s'est déplacer j'jusqu'à la position finale et le bras manipulateur a mouvée à une position finale (voir figure IV.22)

Figure IV.21-- position initiale de Robuter/ULM.



FigureIV.22-- position finale de Robuter/ULM .

IV.8 Conclusion

Dans ce chapitre on a exposer les différents outils utilisés dans l'implémentation de l'architecture proposé, d'abord on s'est focaliser sur le protocole de communication et le modelé client/serveur, ensuite on a défini la manière dont les sockets fonctionne, apres on expliquer fonctionnement des threads et on a présenté les principaux threads introduite dans notre architecture.

En fin on a simulé les modes de notre architecture avec le simulateur turtlesim sur l'interface terminal et on a testé notre architecture réellement sur Robuter/ULM.

Conclusion et perspectives

CONCLUSION GENERALE

Le control d'un robot manipulateur mobile capable de réaliser des tâches est un défi audacieux. De nombreuses capacités doivent être mises à contribution pour relever un tel défi, un robot doit avant tout être capable d'acquérir les informations sur son état et son environnement, connaitre les diffèrent obstacle sités près de lui et prendre une décision et l'appliquer le plus rapidement possible et en temps réel. La solution que nous avons présenté dans ce mémoire répond à ce besoin en coordonnant les différents modules du robot manipulateur mobile et permet une interaction ordonnées des données nécessaires pour le bon fonctionnement du robot.

Dans ce travail, nous avons réussi à mettre en place une architecture de génération de mouvement en permettant l'interfaçage entre l'OS robotique ROS et le système embarqué du Robuter/ULM. Cette architecture est divisé en deux partie : une partie de l'architecture est porté sur la plateforme robotique contenant un pc embarqué sous linux et l'autre est situé sur un pc distant tournant sous l'os Ubuntu / ROS. Cela permet l'usage des nœuds ROS sur cette plateforme. Les deux parties de l'architecture sont relié via une liaison TCP/IP avec un support physique de type wifi.

Toutefois, il et nécessaires de réciser qu'avant d'aboutir à une architecture finale et exploitable

Il nous a fallu de se familiariser avec de nouveaux concepts cruciaux à l'aboutissement de notre projet et faire faces à certain défis :

- La construction des nœuds avec ROS, un outil assez riche.
- Les performances et les ressources réduites t du pc embarqué de la plateforme mobile, l'architecture que nous avons adoptée à l'avantage de déporter tous les calculs et tâche complexe sur un PC distant. De ce fait le robot exécute que les taches simple, e se déplace et réalise ses mission.
- La complexité de fonctionnement de Sydex.
- LA Standardiser DU programme : avoir une plateforme robotique pouvant être contrôlée par un OS robotique utilisé par les grands labos de recherche dans le monde, et ainsi bénéficier des résultats de leur recherche en les adaptant à notre plateforme.
- Utilisation des différents outils proposés par ROS.

- La possibilités d'intégrer de nouveaux périphéries tel que la camera Kinect et ainsi bénéficier de ses avantages technologique dans le domaine de l'imagerie et le suiv

Malgré les résultats concluants obtenus avec cette architecture on n'est pas à l'abri d'une défaillance du au hardware ou au système Syndex, néomoins pour remédier à cela nous avons songé à certain perspectives :

- ❖ Faire des modifications sur le robot afin de pouvoir installer ROS sur cette plateforme.
- ❖ Utilisés la caméra kinect pour la localisation.
- ❖ Utilisé un système de vision pour une correction dynamique de trajectoire et la collaboration avec un système robotisé pour réaliser des tâche nécessitant l'intervention de plusieurs robot.

Annexes

ANNEXE A : Définitions

POO : la programmation orientée objet est un type de programmation qui a pour avantage de posséder une meilleure organisation, surtout dans les gros programmes. Ces derniers seront agencés de façon plus logique et seront donc plus facilement modifiables. La POO est cependant plus difficile à maitriser.

RFLEX : est le nom donné à l'interface de contrôle conçu par la société willow garage pour c'est platforms robotique et qui permet de la contrôler via un menu rudimentaire et un potentiomètre pour la navigation dans derniers.

POSIX : est le nom d'une famille de standards définie depuis 1988 par l'Institue of Electrical and Electronics Engineers (IEEE) formellement désignée par IEEE 1003. Ces standards ont émergé d'un projet de standardisation des API des logiciels destinés à fonctionner sur des variantes du système d'exploitation UNIX.

Le terme POSIX a été suggéré par Richard Stallman, qui faisait partie du comité qui écrivit la première version de la norme. L'IEEE choisit de le retenir car il était facilement mémorisable. Les quatre premières lettres forment l'acronyme de Portable Operating System Interface, et le X exprime l'héritage UNIX de l'Interface de programmation.

IDL : acronyme (d'Interactive Data Language), est un langage de programmation propriétaire apparu à la fin des années 1970, et qui est rapidement monté en puissance dans les domaines de la télédétection et de l'astronomie. IDL est un langage vectoriel de traitement de données et de visualisation très répandu dans l'industrie et dans la recherche.

UML : la description de la programmation par objets a fait ressortir l'étendue du travail conceptuel nécessaire : définition des classes, de leurs relations, des attributs et méthodes, des interfaces etc. pour programmer une application, il faut d'abord organiser ses idées, les documenter, puis organiser la réalisation en définissant les modules et étapes de la réalisation.

C'est cette démarche antérieure à l'écriture que l'on appelle modélisation ; son produit est un modèle. [52]

XML-RPC : est un protocole RPC (Remote procedure call), une spécification simple et un ensemble de codes qui permettent à des processus s'exécutant dans des environnements différents de faire des appels de méthodes à travers un réseau.

XML-RPC permet d'appeler une fonction sur un serveur distant à partir de n'importe quel système (Windows, Mac OS X, GNU/Linux) et avec n'importe quel langage de programmation. Le serveur est lui-même sur n'importe quel système et est programmé dans n'importe quel langage.

ANNEXE B : ROS

Les étapes à suivre pour l'installation de ROS :

Note : le choix de la distribution ROS se fera en fonction de la version de l'os choisi. Pour plus de détaille voir ce lien http://www.ros.org/reps/rep-0003.html

Pour commencer nous allons ajouter les dépôts de ROS dans les sources de paquets de votre système via cette commande :

Sudo sh –c'echo "deb http://packages.ros.org/ros/ubuntu precise main">/etc/apt/sources.list.d/ros-latest.list'

Sudo sh –c'echo "deb http://packages.ros.org/ros/ubuntu precise main">/etc/apt/sources.list.d/ros-latest.list'

Afin de pouvoir installer les parquets de cette nouvelle source, nous allons récupérer la signature de ROS et l'ajouter aux clés connues de notre gestionnaire de paquet

Wget http://packages.ros.org/ros.key-O -| sudo apt-key add-

Maintenant nous pouvons mettre à jour les paquets disponibles :

sudo apt-get update

Installation de ROS desktop full (si vous êtes sur une machine peu puissante, il faudra surement regarder les autres versions)

(http://www.ros.org/wiki/fuetre/Installation/Ubuntu#Installation-1))

sudo apt-get install ros-fuerte-desktop-full

A ce stade, ROS, est installé. Afin de pouvoir s'en servir, il faut un certain nombre de variables d'environnement. Si vous n'avez qu'une version de ROS sur votre machine (théoriquement oui) vous pouvez automatiser l'assignation des variables d'environnement au lancement de votre session.

echo "source/opt/ros/fuerte/setup.bash" »~/.bashre

Nous allons exécuter le script afin d'assigner les variables d'environnement

~/.bashre

Maintenant nous allons installer deux «petits» outils

sudo apt-get install python-rosinstall python-rosdep

Configuration de rosdep

Sudo rosdep init

Rosdep update

Configuration de ROS

L'objectif de cette partie est de mettre en place un « workspace » dans lequel nous pourrons créer nos applications et nos librairies.

Création du dossier « workspace »

mkdir~/fuerte_workspace

Maintenant nous allons initialiser le workspace. Cela consiste à créer des scripts permettant d'ajouter notre workspace dans les variables d'environnement de ROS :

rosws init~/fuerte_workspace/opt/ros/fuerte

Nous allons donc mettre à jour nos variables d'environnement afin de rendre disponible notre workspace

cd~/fuerte_worspace

source setup.sh

A chaque redémarrage, ouverture de session, etc. il faudra mettre les variables d'environnement associées au workspace que vous souhaitez utiliser

Source<path_to_workspace>/setup.sh

Dans notre cas :

source~fuerte_worspace/setup.sh

Cette action peut être automatisée en ajoutant la ligne précédente à la fin du fichier~/bashrc.

Maintenant que nous avons mis à jour nos variables d'environnement, nous pouvons tester la configuration du workspace :

roscd

Pwd

Aperçu sur Linux

Linux est le nom couramment donné à tout système d'exploitation libre fonctionnant avec le noyau Linux. C'est une implémentation libre du système UNIX. Respectant les spécifications POSIX. Ce système est né de la rencontre entre le mouvement du logiciel libre et le modèle

de développement collaboratif et décentralisé via Internet. Son nom vient du créateur du noyau Linux, **Linus Torvald**, c'est un système qui est né dans le milieu Hacker.

Les systèmes basés sur Linux sont majoritairement pour les super-ordinateurs et les Smartphones. Sur les serveurs informatiques, le marché est partagé avec les utres Unix et Windows. Il est largement utilisé comme système embarqué dans les appareils électroniques : télévision, modem, GPS, le système avec toutes ses applications est distribué sous la forme de distributions Linux comme **Slackware, Debian ou Red Hat Entreprise Linux**, la différence essentielle de Linux et les autres systèmes d'exploitation concurrents-comme **Mac OS, Microsoft Windows et Solaris**-réside dans le fait que ce système (Linux) est un système d'exploitation libre, apportant quatre libertés aux utilisateurs, définies par la licence publique générale GNU (**GPL**), les rendent indépendants de tout éditeur et encourageant l'entraide et le partage.

Aperçu sur c+++ :

C++ est l'un des langages de programmation les plus populaires, avec une grande variété de plateformes matérielles et de systèmes d'exploitation.

Le C++ est un langage de programmation permettant la programmation sous de multiples paradigmes comme la programmation procédurale, la programmation orientée objet et la programmation générique.

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello, world!" << endl;
    return 0;
}
```

Le programme commence par une série de déclarations, suivi du code principal. Tout programme doit contenir une fonction « main », qui est le point d'entrée du programme ; des librairies comme « **iostream** » qui est librairie servant aux entrées-sorties, le mot clé «**#include** » est une directive de préprocesseur, qui sert à indiquer que les fonctions utilisées sont décrites dans le fichier « iostream.h »

Chaque instruction est terminée par le symbole « ; » et un bloc d'instruction est délimité par une accolade ouvrante '{'et une autre'}' fermante. L'orienté objet est basé sur l'aspect de classe une classe est donc un objet, qui contiendra des variables et des fonctions, appelées respectivement attributs et méthodes.

Le master comprend une sous-partie très utilisée qui est le Paramètre Server.
Celui-ci, également implémenté sous forme XMLRPC, c'est une sorte de base de données centralisée dans laquelle les nœuds peuvent stocker de l'information et ainsi partager des paramètres globaux.

• **Serveur paramètre** : c'est un serveur qui fait partie du master et qui s'occupe des paramètres (stocke les données dans une localisation centrale).

• **Les topics ou sujets** : l'échange d'information s'effectuer soit de manière asynchrone via un topic ou de manière synchrone via un service.

Un topic est un système de transport de l'information basé sur le système de l'abonnement/publication (subscribe/publish). Ou un ou plusieurs nœuds pour peuveut publier de l'information sur un topic et un ou plusieurs nœuds pourront lire l'information sur ce topic. Le topic est en quelque sorte un bus d'information asynchrone.

Cette notion de bus many –to-many asynchrone est essentielle dans le cas d'un système distribué. Le topic est typé, c'est –à-dire que le type d'information qui est publiée (le message) est toujours structuré de la même manière. Les nœuds envoient ou reçoivent des messages sur des topics.

• **Les messages** : un message est une structure de donnée composite. Un message est composé d'une combinaison de types primitifs (chaines de caractère, booléens, entiers, flottants. . .) et de message (le message est une structure récursive). Par exemple : un nœud représentant un servomoteur du robot, publiera certainement son état sur un topic avec un message contenant par exemple un entier représentant la position du moteur, un flottant représentant sa température, un autre flottant représentant sa vitesse. . .

• **Les services** : le topic ou sujet est un mode de communication asynchrone permettant une communication many –to-many.

Le service en revanche répond à une autre nécessité, celle d'une communication synchrone entre deux nœuds. Cette notion se rapproche de la notion d'appel de procédure distante (remote procédure call –RPC).

• **Les bags ou baguages** : les « bags » sont des formats pour stocker et rejouer les messages échangés. Ce système permet de collecter par exemple les données mesurées par les capteurs et les rejouer ensuite autant de fois qu'on le souhaite afin de faire de la simulation sur des données réelles. Ce système est également très utile pour déboguer un système à posteriori.

La commande « rxbag » permet de visualiser graphiquement les données enregistrées dans le fichier bag.

ANNEXE C : Présentation du manipulateur mobile

Robuter/ULM

1. Informations générales

1.1 A propos du RobuTER

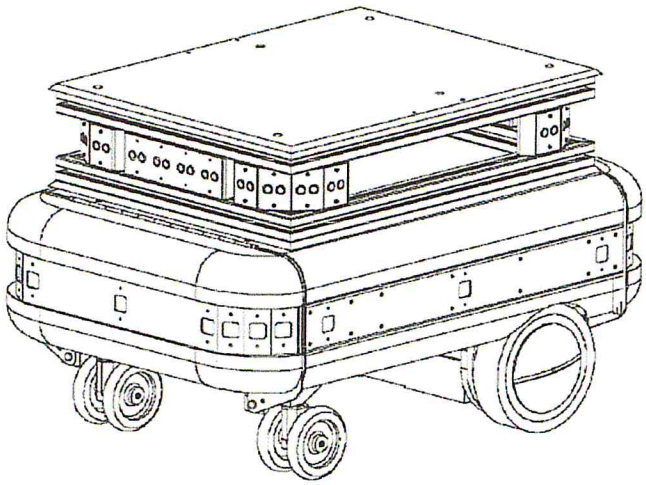


Figure : C.1 vue du RobuTER.

Le RobuTER est une plateforme automatisée et programmable de transport d'objets de taille moyenne (jusqu'à 150kg). Ses 2 roues motrices permettent une grande mobilité sur sols lisses. Grace à son pilotage en différentiel de vitesse, le RobuTER peut tourner sur lui –même.

Dans la version proposée, il est équipé d'un bras Ultra-Léger.

1.2 A propos du Bras Ultra Léger

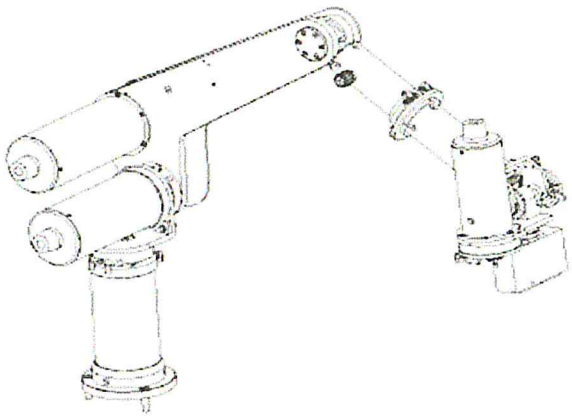


Figure C.2- Vue du Bras Ultra Léger.

Pesant 20kg, le bras Ultraléger est un manipulateur peu onéreux, prêt à l'emploi, pouvant être utilisé sur une plateforme mobile. Il est aussi bien adapté pour le transport ou la manipulation de petites pièces, que le ramassage d'échantillon au sol.

L'ensemble de la plate-forme munie du bras est contrôlé par un PC embarqué (Pentium MMX 233 MHz sous Linux à et 3 contrôleurs cb 555 fabriqués par ROBOSOFT. Le premier cb555 se situe à l'intérieur de la base mobile, tandis que les 2 autres sont installés dans le contrôleur du bras. Cette architecture de contrôle présente l'avantage pour les utilisateurs de programmer les mouvements de la plateforme soit en utilisant les outils de développement orienté « haut niveau », soit en utilisant la chaine de compilation dédiée **SynDEx** pour l'accès au « bas niveau ».

Les caractéristiques de l'ensemble sont les suivantes :

- 2 roues motrices à bandage de polyuréthane
- Direction par différentiel de vitesse
- Bras manipulateur à 6 axes avec pinces

2. Spécifications :

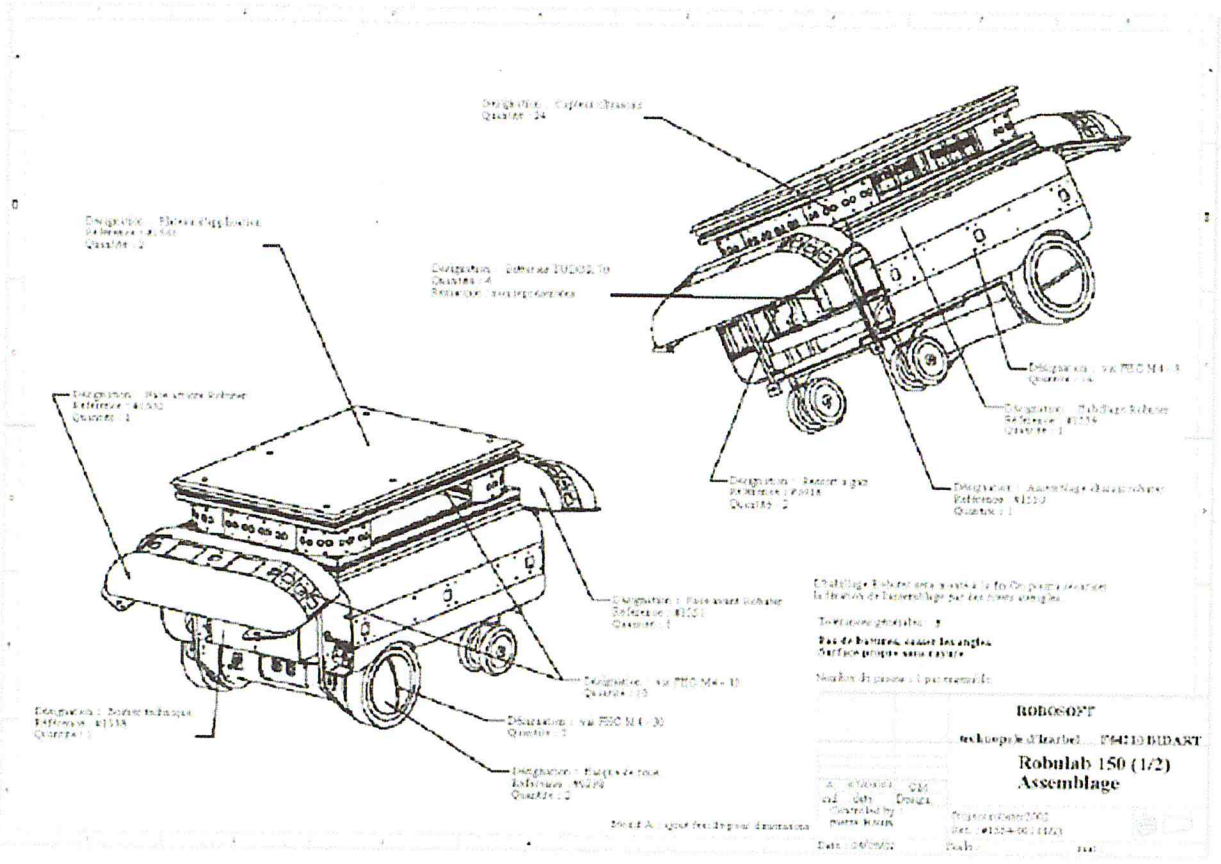


Figure : C.3 Detail du RobotER.

2.1 RobuTER

2.1.1 Architecture de la plate-forme

Le RobuTER est une plate-forme à 2 roues motrices dune capacité de charge de 15KG.

Les roues sont actionnées par des moteurs électriques à courant continu et lui permettent d'atteindre une vitesse nominale de 2,6 m/s, avec des roues de 250 mmde diamètre, et un couple nominal de 22 Nmpar roue.

La direction du RobuTER est donnée par le différentiel de vitesse des 2 roues. Les 2 roues folles à l'avant de la plate-forme assurent la stabilité de l'ensemble. La consommation nominale est de 30A entre 30 et 48 VDC, le courant de crête est de 60A pendant 2s à 48VDC.

2.1.2 Moteur

- Motoréducteur : MBR CF80.35F2
- Vitesse de définition : 3000 tr/min
- Masse du moteur : 2,5 kg
- Tension d'alimentation de définition : 48VDC
- Courant permanent en rotation lente : 15A
- Couple permanent en rotation lente : 22N.m
- Courant crête : 30A
- Rapport de réduction : 1/15

2.1.3 Codeurs

- Codeur incrémental line driver
- Nombre de points : 500
- Masse : 0.085 kg

2.1.4 Capteurs Ultrasons

Le robuTER dispose également d'une ceinture d'ultrasons. Ce paragraphe propose de donner la position (x, y, z, théta) de chaque capteur sachant que le premier est celui qui se trouve à gauche de la ceinture avant lorsqu'on se met dans le sens du robuTER.

3. Guide de démarrage rapide

3.1 Connecteurs et panneau de contrôle

L'interrupteur principal se trouve à la gauche du RobuTER, il sert à allumer ou éteindre la plate-forme. Soulevez l'habillage arrière pour y accéder. Le panneau de contrôle à l'arrière du RobuTER présente les éléments suivants :

- Indicateur du niveau de batterie : une barre lumineuse composée de dels indique le niveau d'énergie restant. L'affichage digital, en dessous des dels, indique le temps global de fonctionnement de la plate-forme depuis la première mise en route.

Il est fortement recommande de ne pas utiliser la plate-forme lorsque le niveau des batteries est inférieur à 30%. Les batteries doivent être mises en charge immédiatement.

- Connecteur de charge : connecter le chargeur ici pour recharger les batteries. Le chargeur doit être connecté aussi souvent que possible.
- Connecteur de joystick : connecter ici le joystick analogique.
  Il est important de connecter le JOYSTICK avant de démarrer la plate-forme en mode manuel. Sans le joystick les mouvements de la plate-forme seront imprévisibles.
- Arrêt d'urgence ce bouton sert à arrêter la plate-forme en cas d'urgence. L'arrêt d'urgence doit être déclenché dans toutes les situations anormales. Une fois l'arrêt d'urgence déclenché, le bouton reste enfoncé, tournez- le dans le sens des aiguilles d'une montre pour le déverrouiller.

3.2 Branchements du PC :

Le PC embarqué possède les connecteurs suivants :

- 2 ports CAN (carte Adlink "7841 dual-CAN " intégrant 2 chipsets SJA1000)
- 2 ports series RS232

3.3 Démarrage de la plate-forme

La plate-forme est livrée avec le mode d'opération base sur SynDEX. Pour démarrer la plate-forme, les instructions suivantes doivent être opérées :

- Allumer la plate-forme en tournant le sectionneur général (le bouton d'arrêt d'urgence doit être enfoncé).
- Relâcher le bouton d'arrêt d'urgence (tourner dans le sens des aiguilles d'une montre) afin d'alimenter les contrôleurs.
- Attendre 2 secondes, un claquement provenant des relais devrait se faire entendre.
- Démarrage de **SynDEx** :

Allumer le PC embarqué et attendre la fin du démarrage de Linux. Lorsque le prompt de login s'affiche, entrer le nom « root » et le mot de passe « robuter0 ». Une fois loggé, aller dans le répertoire « syndex » qui contient les applications pour opérer la plate-forme. Ce répertoire contient différents morceaux de logiciels de contrôle de la plate-forme, la documentation sur les macros SynDEX pour le cb5555 et sur les macros linuxIO_se trouve également dans ce répertoire (dans le sous répertoire «doc »). Ces macros sont utilisées pour l'interface entre les APIs SynDEx et les programmes en C/C++ s'exécutant dans l'espace utilisateur de Linux.

```
Red Hat Linux Release 8.0 (Psyche)
Kernel 2.4.18-rthal5 on a i586

robuter0 login: root
password: ******

[root@robuter0 /root]# cd syndex
[root@robuter0 syndex]#
```

Figure C.4-terminal de login

Changer pour le répertoire « robuter-Ularm ». Ce répertoire contient les fichiers sources SynDex de l'application de control manuel de la plateforme. Les utilisateurs avancés souhaitant développer leur propre application SynDex de control sont fortement encouragées à commencer par l'application fournie afin de comprendre le fonctionnement interne du RoboTER. Voici maintenant l'instruction pour démarrer l'application de control manuel (téléchargement) :

- Réarmer le contrôleur en enfonçant puis relâchant le bouton d'arrêt d'urgence.
- Taper au clavier « make clean make »
- Des messages concernant la compilation et le chargement apparaissent.
- Un claquement correspondant au relâchement des freins devrait se faire entendre.
- Charger l'application utilisateur pour choisir le mode d'opération : le programme « demo-0.1 » est utilisé pour rédigé le RoboTER et le bras à l'aide

du joystick, les programmes « demo-0.3 » sont utilisés pour des démonstrations avec le bras Ultra Leger.

- Choisir les actionneurs à contrôler en tapant au clavier les caractères correspondants :

r	RobuTER
1	Axes 1 et 2 du bras
2	Axes 3 et 4 du bras
3	Axes 5 et 6 du bras
g	Pince (gripper)

- Une fois que l'application est lancer, 2 valeurs entiere difilent constamment à l'ecran (commen indique sur la figure). Elles correspondent aux valeurs Xet Y du joyistick.
- Le joyistick contrôle maintenant les axes précedemment selectionnées.

```
****** RTAI NEWLY MOUNTED (MOUNT COUNT 1) ******

==== RT_memory_manager_v1.3_Loaded. ====

***** STARTING THE UP REAL TIME SCHEDULER WITH LINUX *****
***** FP SUPPORT AND READY FOR A PERIODIC TIMER *****
***<> LINUX TICK AT 100 (HZ) <>***
***<> CALIBRATED CPU FREQUENCY 233871000 (HZ) <>***
***<> CALIBRATED TIMER-INTERRUPT-TO-SCHEDULER LATENCY 2689 (ns) <>***
***<> CALIBRATED ONE SHOT SETUP TIME 2009 (ns) <>***

Probed CAN base address: 0xE000
Probed CAN IRQ: 0xB

Downloading processor ID 0x4000 station!... loaded.
Downloading processor ID 0x4001 station!... loaded.
Downloading processor ID 0x4002 station!... loaded.
```

Figure : C.5 affichage de l'application.

Voici maintenant la séquence d'instruction utilisé pour arrêter l'application du RoboTER.

Pour arrêter l'application, presser d'abord les touches « Ctrl+C » afin de forcer l'arrêt de l'application en C. entré ensuite l'instruction « make stop ». Le noyau va ensuite afficher des messages indiquant le retrait correct des modules de l'espace-temps-réel du noyau. La plateforme est maintenant prête pour un nouveau téléchargement.

Bibliographie

Bibliographie

[1] Isaac Asimov , Les robots (I, robot) Manuel de la Robotique 58e édition (2058 après J.-C.) 1950 .

[2] David Filliat. Robotique Mobile. Engineering school. Robotique Mobile, ENSTA ParisTech, 2011, pp.175.

[3] Nilsson, Nils J. **Shakey The Robot**, Technical Note 323. AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Apr 1984.

[4] H.Saski,T.Takahashi,E.Nakano,"A door opening method by a mobile manipulator with passivejoints".Journal of Robotic Society of japan,19(2),pp.129-136,2001.

[5] Y.Chen,L.Liu,MZhang,H.Rong."Study on Coordinated Control and Hardware System of a Mobile Manipulator".Proceedings of the 6th world Congress on Intelligent Control and Automation,Dalian,China.pp.9037-9041,June 21-23,2006.

[6]E.Beaudry, Planification de taches pour un robot mobile autonome, Sherbrooke, Québec,

Canada aout 2006.

[7]Akli Isma,Elaboration d'une stratégie de coordination de mouvements pour un manipulateur mobile redondant,mémoire Mgister,USTHB ,juillet 2007.

[8] D.Fox,W.Burgard et S.Thrun : Markov localization for mobile robots in dynamic environments.Journal of Artificial Intelligence Research (JAIR),11 :391-427,1999.

[9] S.Thrun,D.Fox.et W.Burgrad :Robust Monte Carlo localization for mobile Robots.Artificial Intelligence,128(1-2):99-141,2001.

[10]R.C Gonzalez et R.E. Woods :Digital Image Processing .Addison-Wesley Long-man Publishing Co,Inc ,Boston,MA,USA,2001.

[11]E.Trucco et A.Verri : Introductory Techniques for 3-D Computer Vision.Pren-tice Hall PTR,Upper Saddle River,NJ,USA,1998.

[12]D.Létourneau,F.Michaud et J.-M.Valin :Autonomous robot that can read .Dans EURASIP Journal on Applied Signal Processing,Special Issue on Advances in intelligent Vision Systems:Methods and Applications,vol.3,pages 340-361,2004.

[13] Xavier BROQUERE : Planification de trajectoire pour la manipulation d'objets et l'interaction Homme-robot,Université de Toulouse,France ,5 juillet 2011.

[14] Benbouali Riadh ,el amine Ouzzane ,architecture de control multi-agents pour les robots manipulateurs mobiles,mémoire Magister, USTHB ,2009.

[15] David FILLAT,Robotique mobile école Nationale de techniques Avancées Paris Tech ,octobre 2011.

[16] http://www.humanoides.fr

[17] Salah KERMICHE , Modélisation et commande d'un robot par méthodes intelligentes , DOCTORAT D'ETAT ,UBMA,2006.

[18] D. Hunziker, M. Gajamohan, M. Waibel, and R. D'Andrea. Rapyuta : The roboearth cloud engine. In Robotics and Automation (ICRA), 2013 IEEE International Conference on, pages 438–444. IEEE, 2013

[19] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros : an open-source robot operating system. In ICRA workshop on open source software, volume 3, 2009.

[20] G. K. Kraetzschmar, H. Utz, S. Sablatnög, S. Enderle, and G. Palm. Miro - middleware for cooperative robotics. In RoboCup 2001 : Robot Soccer World Cup V, pages 411–416, London, UK, 2002. Springer-Verlag

[21] K. Johns and T. Taylor. Professional Microsoft Robotics Developer Studio. Wrox Press Ltd., Birmingham, UK, UK, 2008.

[22] C. Côté, Y. Brosseau, D. Létourneau., C. Raïevsky, Y. Brosseau, and F. Michaud. Using marie for mobile robot component development and integration. Software Engineering for Experimental Robotics Book Series Springer Tracts in Advanced Robotics Publisher Springer Berlin / Heidelberg, 30/2007 :211–230, April 2007.

[23] Stefan-Gabriel Chitic, Julien Ponge, Olivier Simonin. Intergiciels pour systèmes multi-robots: état de l'art. UbiMob2014 : 10`emes journées francophones Mobilité et Ubiquité, Jun 2014, Sophia Antipolis, France. 8 p., 2014.

[24] http://www.dicofr.com

[25] Jérome Laplace,Présentation de ROS,Dévloppez-corn,Génération Robots;Nov 2011.

[26] A.Valero,G.Randelli,F.Botta,D.Rodriguez-Losada, M Hernado Operator Performance in Exploration Robotics Joournal of Intelligent & Robotic Systems,1—21,2011.

[27] Jonathan Bohren,ROS Basic Concepts ,Institue for Software Technology,Austria,2012.

[28] Jonathan Bohren ,ROS Basic Crash-Course,Part 1 :Introduction to ROS distribution ,build system and infrastructure,Laboratory for Computational Sensing + Robotics :Johns Hopkins University ,2010.

[29] DeEick Bommarito ,ROS ,Multi-Disciplinary Robotics Club /Rochester Institute of Technology,UK,2012.

[30]Nayden Chivarov ,Robot Arm Control,under ROS/Ubuntu,Institute of System Engineering and Robotics,Sofia,2010,Bulgarie.

[31]Lorenz Mosenlechner ,An Introduction to ROS ,Inteligent Autonomus Systems/AIS

University of Freiburg ,Germany,2011.

[32] Jurgen Hess,Felixx Endress, Armin Homung Bastien Steder ,Jurgen Sturm,Open Source Robot Operating System,Intelligent Autonomus System/AIS,University of Freiburg,Germany,2011.

[33] Arnaud Degroote. Une architecture de contrôle distribuée pour l'autonomie des robots. Robotique [cs.RO]. Institut National Polytechnique de Toulouse - INPT, 2012. Français

[34] N.Achour,R.Toumi Modélisationd'un manipulateur mobile –Application à une tache d'écriture ,International Conferance of Electronic Technologies of information and Telecommunications ,SETIT ,2007.

[35] Nicolas Guérineau, Clément Moignard, Pierre Pomiers , RobuTER et Bras Ultra Léger Manuel d'utilisation et de maintenance , 24/12/2004 ,version : 0.4.

[36] BEAUDRY, Julien, Projet SpinoS: Conception et contrôle d'un robot mobile à vitesses différentielles -- Projet de fin d'étude, Montréal, École Polytechnique de Montréal, 2001.

[37] HILL.J;PARK.W.T.real time control of robot with a mobile camera .9th Int. Symp on Industrial Robots,pp:233-246,1979.