الجمهورية الجزائرية الديمقراطية الشعبية
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
جامعة سعد دحلب–البليدة–
Université Saad Dahlab– Blida–
Faculté des sciences

C.D.T.A

MEMOIRE

Présenté pour l'obtention du **diplôme** de **MASTER**

En : Informatique

Spécialité : Génie des systèmes informatiques

Sujet

Planification automatique et hiérarchique des taches Pour les robots manipulateurs mobiles.

Réalisé par :

- Ould Ahmed Mohamed Taleb.
- Elmetennani Mounir.

Dirigé par :

- Mme.Akli Isma.
- M. kameche Abdallah.

2016/2017

Remerciements :

On tient, au terme de ce travail, à présenter nos vifs remerciement à dieu qui nous as donne la force pour accomplir ce travail à tous les personnes qui ont contribué, de près ou loin, à son bon déroulement

On tient à présenter tous nos respects et notre gratitude à Mr. pour nous avoir offert l'opportunité d'effectuer ce stage, ainsi pour son suivi et encouragement tout au long de ce travail.

Notre gratitude s'adresse également à Madame Akli et Mr.kameche pour leur encadrement et pour l'aide qu'ils m'ont prodigué durant ce rapport, ainsi qu'a tous mes formateurs.

On remercie également les membres des jurys qui ont accepté d'évaluer mon travail.

Dédicace :

Nous dédions ce travail à :

Nos mères, sources de tendresse et d'amours pour leurs soutiens tout le long de notre vie scolaire.

Nos pères, qui nous ont toujours soutenus et qui ont fait tout possible pour nous aider.

Nos frères et nos sœurs, que nous aimons beaucoup.

Notre grande famille.

Nos cher ami (e) s, et enseignants.

Tout qu'ont collaboré de près ou de loin à l'élaboration de ce travail.

Que dieu leur accorde santé et prospérité

Mohamed et Mounir.

Résumé :

Les robots manipulateurs mobiles se présentent, en général comme des bras manipulateurs poly-articulés (assurant des tâches de saisie d'objets avec des pinces), portés par des véhicules à roues. Ces robots offrent la possibilité d'exécuter des tâches complexes combinant la mobilité et la manipulation (maintenance industrielle, robotique de service, etc.). La planification de tâches s'avère être une étape nécessaire à cause du degré de complexité relativement élevé des tâches pouvant être accomplies. Les planificateurs de tâches sont des modules se situant à un haut niveau décisionnel.

Le but du sujet proposé est la conception et l'implémentation d'une architecture hiérarchique dédiée à la planification et à l'exécution de tâches à réaliser par un robot manipulateur mobile. La planification s'avère indispensable puisqu'il est nécessaire de décomposer une tâche complexe à exécuter.

Dans notre cas on a opté pour la planification hiérarchique (HTN) afin de décomposer deux tâches complexe : Saisie d'un objet et ouvrir une porte. La décomposition a été pensée pour le robt Robuter ULM.

Mots clés : Planification , HTN, Robot mobile, Robuter ULM, ROS.

Abstract:

The mobile manipulator robots are generally articulated manipulators (for gripping objects with grippers), carried by wheeled vehicles. These robots offer the possibility of performing complex tasks combining mobility and manipulation (industrial maintenance, service robotics, etc.). Task planning is a necessary step because of the relatively high complexity of tasks that can be accomplished. Task planners are modules at a high decision-making level.

The aim of the proposed subject is the design and implementation of a hierarchical architecture dedicated to the planning and execution of tasks to be carried out by a mobile manipulator robot. Planning is essential because it is necessary to decompose a complex task to be executed.

In our case, we used Hiearchical Task Network (HTN) planning in order to make our robot execute two tasks : holding an object and opening a door . This decomposition was thought for Robuter ULM.

Keys words : Planning , HTN, mobile robot, Robuter ULM, ROS.

Table des métiers

Chapitre 3 – Conception

Chapitre 4 – Implémentation

Liste des figures

Chapitre 1 : Généralités sur la robotique

Planificateurs de taches pour robots manipulateur mobile

Chapitre 4 : Implémentation

Introduction générale

Introduction générale

Depuis fort longtemps, l'humain rêve de créer des machines intelligentes capables d'effectuer des tâches à sa place. Ainsi, les humains auraient plus de temps à consacrer pour leurs loisirs, ou prendraient moins de risques pour effectuer des tâches dangereuses. Or créer une machine pouvant réaliser des tâches que seuls les humains sont normalement capables de faire n'est pas aussi simple qu'on pourrait le penser. En effet, sans toujoursy penser, les tâches les plus élémentaires de la vie quotidienne d'un humain peuvent devenir extrêmement complexes lorsqu'on les analyse de plus près. Par exemple, participer à un congrès ou à une conférence peut devenir un véritable casse-tête. Avant tout, il faut consulter son agenda afin de s'assurer de sa disponibilité. En présence de conflits d'horaire, il est alors nécessaire de négocier des arrangements avec les personnes concernées.

Une fois cette étape franchie, il faut s'inscrire et payer les frais d'inscription. Par la suite, il faut se rendre sur les lieux de l'événement en combinant divers moyens de transport comme l'automobile, l'autobus et peut-être même l'avion. Une fois arrivé sur place, il faut repérer le kiosque d'enregistrement et s'y présenter afin d'obtenir son porte-nom ainsi que tous les autres documents pertinents, comme le programme de l'événement. De plus, tout au long du congrès ou de la conférence, il faut se rendre dans les bonnes salles, et ce, aux bons moments. Bien que ces tâches semblent simplespour nous les humains, cela est loin d'être aussi évident pour un robot. Afin d'être autonome, un robot 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, un robot possède des capteurs, comme des sonars et un dispositif à balayage laser servant à mesurer des distances entre lui-même et les obstacles 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 efficaces et sécuritaires afin d'éviter des collisions avec les obstacles. De plus, un robot est souvent appelé à communiquer avec les gens ou d'autres agents situés à proximité. Cela peut être fait de différentes façons, comme par

communication vocale ou depuis une interface graphique. En plus de pouvoir percevoir globalement son environnement, 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 les images acquises par la ou les caméras installées sur le robot. Après avoir identifié etlocalisé un objet, on peut imaginer que le robot ait ensuite à manipuler cet objet avec son bras-robot. Bien que beaucoup d'autres capacités de perception et d'action pourraient être ajoutées à cette liste, l'important est d'avoir à l'esprit que la robotique mobile est un domaine hau-tement multidisciplinaire qui fait l'objet de beaucoup de recherches dans des disciplines très diversifiées.

Pour cette raison, dans le présent mémoire, nous faisons abstraction de la plupart de ces capacités robotiques et nous considérons avoir accès à celles-ci puisqu'ils découlent de d'autres travaux, indépendamment de ceux présentés ici.

Enfin, une autre capacité robotique, tout aussi importante que celles énumérées précédemment, est la capacité pour un robot de prendre lui-même ses propres décisions pour réaliser et coordonner des missions complexes. Cette capacité est d'une grande importance puisque, pour beaucoup de tâches robotiques, il peut exister de nombreuses façons de les réaliser. Par raisonnement, le robot doit sélectionner les meilleures actions à effectuer pour réussir adéquatement sa mission.

De plus, dans certaines missions robotiques, il est impératif de bien séquencer les tâches afin de respecter diverses contraintes permettant au robot d'avoir un comportement cohérent et efficace. Par exemple, pour des tâches de livraison de colis, un certain ordre d'exécution doit être respecté : il faut aller à l'endroit decueillette avant de saisir l'objet et ensuite se rendre à l'endroit de livraison avant de le déposer. Il peut également arriver qu'un robot ait à exécuter des tâches à l'intérieur d'une certaine fenêtre de temps.

Au fil des ans, plusieurs solutions ont été proposées afin de donner aux robots la capacité de coordonner leurs missions. L'une d'entre elles est l'utilisation de techniques de planification dans le domaine de l'intelligence artificielle. Dans le présent mémoire, nous nous concentrons quasi exclusivement sur ce sujet. Nous étudions plus précisément la planification et la coordination de missions pour des robots mobiles et intelligents

évoluant dans des environnements dynamiques. Nous proposons un module de planification proactive spécialement adapté à la robotique mobile, et nous accordons une attention particulière à son intégration dans une architecture robotique.

宋es travaux présentés dans ce mémoire répondent à cette problématique précise, soit

de donner des capacités de planification à Router ULM, un robot dont dispose le laboratoire du CDTA. En premier lieu, nous présentons notre planificateur, un nouvel algorithme de planification mieux adapté à cette problématique. Dans une seconde étape, nous élaborons un domaine de planification, destiné à ce nouveau planificateur.

Ce mémoire est organisé de la façon suivante :

- Le premier chapitre introduit des notions de base en robotique mobile.
- Le chapitre 2 présente les algorithmes de planification a usage général et les algortihmes de planification dédiés au domaine de la robotique.
- Le chapitre 3 décrit notre conception et les fonctions complexes sur lequelles on va travailler .
- Le chapitre 4 contient les résultats qu'on a obtenu en apliquant notre solution.

Chapitre 1

Généralités sur la robotique

1.1 Introduction :

De nos jours avec l'évolution de la technologie, la notion de la–robotique devient familiarisée dans notre vie quotidienne. Avant d'entrer dans le vif du sujet qui concerne la manipulation de tâches pour un robot manipulateur mobile, il est primordial d'avoir une idée générale sur la robotique, et plus précisément sur les robots manipulateurs mobiles qui nous ramènent vers la robotique de service afin de comprendre les interactions entre les différents modules auxquels nous faisons référence

1.2 Définition de la robotique :

La robotique peut être définit comme l'ensemble des Techniques et études tendant à concevoir des systèmes Mécaniques, informatiques ou mixtes, capables de se substituer à L'homme dans ses fonctions motrices, sensorielles et intellectuelles [1].

1.3 Définition d'un robot :

a. Un robot est définit comme étant un appareil automatique capable de manipuler des objets, ou d'exécuter des opérations selon un programme fixe ou modifiable [2].

b. Un robot est définit comme étant un système mécanique de type manipulateur commandé en position reprogrammable, polyvalent à usages multiples, à plusieurs degrés de liberté, capable de manipuler des matériaux, des pièces, des outils et des dispositifs spécialisés, au cours de mouvements variables et programmés pour l'exécution d'une variété de tâches Il a souvent l'apparence d'un, ou plusieurs, bras se terminant par un poignet. Son unité de commande utilise, notamment, un dispositif de mémoire et éventuellement de perception et d'adaptation à l'environnement et aux circonstances [3].

Ces machines polyvalentes sont généralement étudiées pour effectuer la même fonction de façon cyclique et peuvent être adaptées à d'autres fonctions sans modification permanente du matériel [4].

c. Tout comme les machines-outils à commande numérique, dont ils empruntent la technologie, les robots sont des matériels programmables, donc polyvalents. Pilotés à l'aide de systèmes informatiques, ils peuvent être programmés pour exécuter des tâches diversifiées, contrairement à l'idée répandue, mais inexacte, du travail robotisé [5].

1.4 Historique :

L'évolution de la robotique est passée par plusieurs étapes. Le premier robot industriel a été installé en 1961 dans une fonderie d'aluminium américaine, depuis la robotique est devenue une réalité industrielle avec plus de 200 000 robots en service dans le monde [6].

a. 1947 : Premier manipulateur électrique télé-opéré.
b. 1954 : Premier robot programmable.
c. 1961 : Utilisation d'un robot industriel, commercialisé par la société UNIMATION (USA), sur une chaîne de montage de General Motors.
d. 1961 : Premier robot avec contrôle en effort.
e. 1990 : la robotique s'intègre dans le concept de CFAO acronyme de conception et fabrication assistée par ordinateur.
f. Actuel : tels que humanoïdes. Un robot humanoïde est définit étant un robot dont l'apparence générale rappelle celle d'un corps humain. Généralement, les robots humanoïdes ont un torse avec une tête, deux bras et deux jambes, bien que certains modèles ne représentent qu'une partie du corps, par exemple à partir de la taille. Certains robots humanoïdes peuvent avoir un « visage », avec des « yeux » et une « Bouche » [7].

1.5 Classification de robot :

Dans la littérature, les robots sont classés selon deux catégories : il existe la classification par type et la classification par génération **[8]**.

1.5.1 Par type :

a) Les manipulateurs :

- Les trajectoires sont non quelconques dans l'espace
- Les positions sont discrètes avec 2 ou 3 valeurs axe.
- La commande est séquentielle

b) les télémanipulateurs :

Appareils de manipulation à distance (pelle mécanique, pont roulant),
Apparus vers 1945 aux USA:

- Les trajectoires peuvent être quelconques dans l'espace
- Les trajectoires sont définies de manière instantanée par l'opérateur, Généralement à partir d'un pupitre de commande (joystick).

c) les robots :

- Les trajectoires peuvent être quelconques dans l'espace.
- L'exécution est automatique, les informations. Extéroceptives peuvent modifier le comportement du robot.

d) Les robots didactiques :

Qui sont des versions au format réduit des précédents robots. La technologie est différente, de même que les constructeurs. Ils ont un rôle de formation et d'enseignement, ils peuvent aussi être utilisés pour Effectuer des tests de Faisabilité d'un poste robotisé.

e) Les robots mobiles autonomes : Les possibilités sont plus vastes, du fait de Leur mobilité. Notamment, ils peuvent être utilisés en zone dangereuse (Nucléaire, incendie, sécurité civile, déminage), inaccessible (océanographie, Spatial).De tels robots font appel à des capteurs et à des logiciels Sophistiqués. On peut distinguer deux types de locomotion : Les robots Marcheurs qui imitent la démarche humaine, et les robots mobiles qui Ressemblent plus à des véhicules.

1.5.2 Par génération : d'après les études récentes on distingue trois générations de robots :

a) Le robot passif : Il est capable d'exécuter une tâche qui peut être complexe, mais De manière répétitive, il ne doit pas y avoir de modifications intempestives de L'environnement. L'auto-adaptabilité est très faible, de nombreux robots Sont encore de cette génération [7].

b) Le robot actif : Il devient capable d'avoir une image de son environnement, et Donc de choisir le bon comportement (sachant que les différentes Configurations ont été prévues). Le robot peut se calibrer tout seul [7].

c) Le robot intelligent : Le robot est capable d'établir des stratégies, ce qui fait Appel à des capteurs sophistiqués, et souvent à l'intelligence artificielle [7].

1.6 La robotique de service :

Les études récentes concernant l'impact de la robotique dans la société ont clairement établi qu'il sera nécessaire d'utiliser des robots pour effectuer dans un proche avenir des tâches de service et d'aide dans les domaines industriels, de l'inspection en milieux difficiles et dangereux, de sécurité et de surveillance et de service aux personnes âgées et/ou dépendantes. Les chercheurs imaginent qu'un robot capable de se déplacer dans un environnement initialement conçu pour les humains et capable de réaliser des tâches de service à domicile, est une solution économiquement très intéressante pour répondre aux besoins.

La mise au point d'un tel robot–répond dès lors, non seulement à ce problème d'aide et de service à la personne, mais permet également l'utilisation de ce même robot pour des applications de surveillance et d'intervention en milieux dangereux ou en situation de catastrophes [8].

1.7 Domain d'application :

Du fait de la diversité croissante de projet et de modèles, il est difficile de faire une classification des robots de service. Pour cela il est préférable d'illustrer les domaines d'application par des exemples [9].

1.7.1 Les robots d'intervention : La robotique d'intervention (traduction de Field robotics) n'est pas – globalement – au cœur de nos travaux ; elle est cependant souvent incluse dans la robotique de service dans la littérature. Les robots d'intervention sont en général des robots télé opérés (opérés à distance) par des commandes directes (joysticks, bras maître et autres organes de commande physiques ou virtuels), ou en semi-autonome par des ordres de haut niveau pour réaliser et enchaîner des tâches (« va à telle position », «prends cette pièce désignée sur un écran »)…

L'opérateur est donc systématiquement dans la boucle de commande du robot afin d'interagir et l'aider à réaliser sa mission en fonction des évolutions de l'environnement matériel et humain. Les robots d'intervention sont en général utilisés pour remplir des tâches dans des environnements difficiles d'accès ou dangereux pour les humains, ou encore lorsque l'absence d'humain rend l'exploitation plus aisée ou plus efficace.

1.7.2 Les robots de défense : Les robots militaires entrent pour la plupart dans le champ des robots d'intervention. Ils portent des fonctions de reconnaissance, de surveillance, de déminage ou de destruction. La décision de tir sur les robots armés est pour l'instant, pour des raisons d'éthique, sujette à la décision de l'opérateur humain.

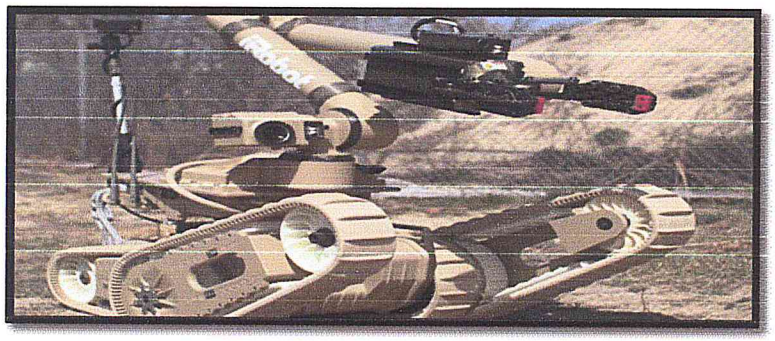


Figure 1. 1 Robot de déminage Packbot 510 EOD

1.7.3 Sécurité civile : La robotique de sécurité civile est mise en œuvre notamment lors des interventions sur catastrophes naturelles. Des robots d'exploration, Notamment, permettent d'explorer des lieux inaccessibles à la recherche de victimes. Dans ce champ, on trouve également les robots utilisés par les forces de l'ordre pour le désamorçage ou la destruction des colis piégés ; dans ce dernier cas, la qualification de « Robot » pourrait cependant être discutée à la lumière de La définition que nous proposons.

- Nucléaire : Le nucléaire est un domaine précoce de développement de la robotique d'intervention. Notamment, il semble que l'expertise initiale du CEA sur la robotique se soit construite sur ce type d'application. Elle concerne essentiellement l'intervention en environnement irradié, nocif pour l'opérateur humain. Elle a donné lieu au développement de technologies « durcies » pour résister à des niveaux plus ou moins élevés de rayonnement

1.7.4 Exploration spatial : Les robots sont aujourd'hui le moyen privilégié de L'exploration spatiale. Les missions vers Mars en sont un exemple significatif : les Délais de transmissions de l'information n'autorisent pas une télé opération en Temps réel par un manipulateur distant (sur Terre). Ce type d'applications est Porteur d'innovations importantes en termes de perception, de locomotion, de Robustesse, etc.

1.8 Le robot manipulateur mobile :

1.8.1 Définition : Comme son nom l'indique le robot manipulateur mobile est Constitué de deux sens le robot manipulateur et le robot mobile voir figure 1.2 :

1.8.1.1 Le robot manipulateur : Des robots ont une base fixe afin de réaliser une tâche précise ou répétitive. Par exemple : les robots industriels et les robots pour l'assistance médicale.

1.8.1.2 Le robot mobile : Sont des robots capables de se déplacer dans son environnement, les robots mobiles à roues constituent le gros des robots mobiles car ils sont plus simples à réaliser que les autres types. Par exemple : les robots explorateurs, robots de service, robots ludiques.

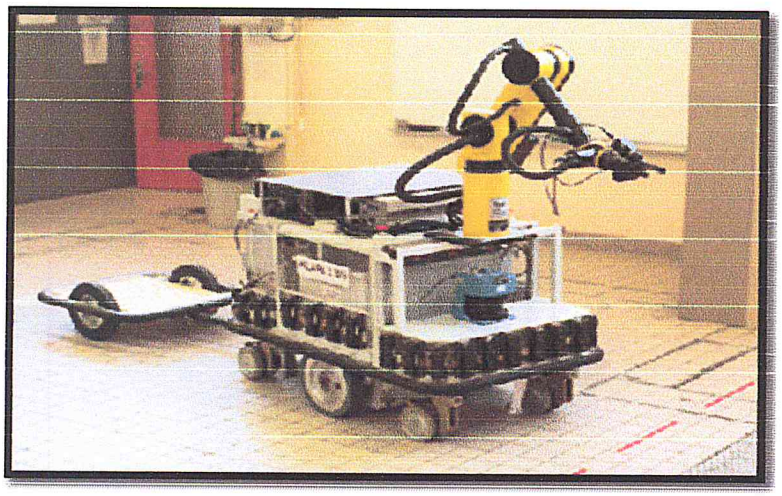


Figure1.2 Robot mobile H2 bis est son bras manipulateur

1.8.2 Composant d'un robot manipulateur mobile : un robot manipulateur mobile est constitué de composantes matérielle et logicielle. Parmi les composantes matérielles

on peut citer la plate-forme mobile .Par contre les composant logicielle sont généralement : les modules de vision, de localisation, de navigation et de séquencement de données.

1.8.2.1 Manipulateur : c'est le corps principal du robot qui comprend les fonctions, les articulations, et d'autres éléments de structure du robot. Il convient de noter ici que le manipulateur seul n'est pas un robot [9]. La figure 1.3 illustre le manipulateur.

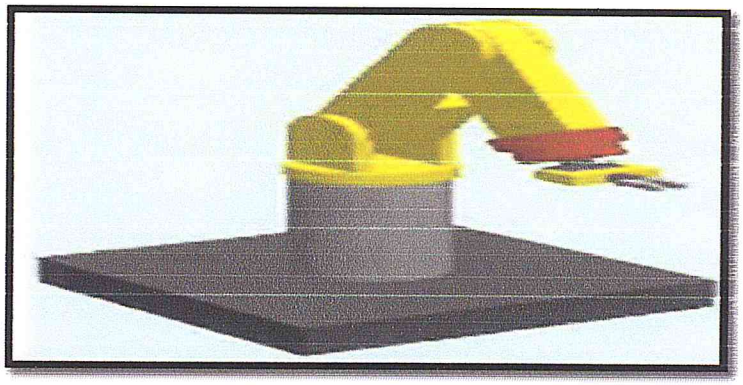


Figure 1.3 le manipulateur

1.8.2.2 Effecteur finale : cette partie est reliée à la dernière jonction (main) d'un manipulateur qui gère généralement les objets, établit des connexions à d'autres machines ou effectue les tâches requises [9].la figure 1.4 illustre l'effecteur finale.

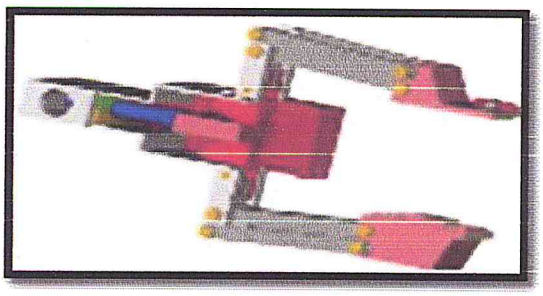


Figure 1.4 effecteur finale d'un manipulateur.

1.8.2.3 Actionneurs : Pour bouger à l'intérieur de son environnement et interagir avec celui-ci, un robot est équipé d'actionneurs. Par exemple, un robot est muni d'un ou de plusieurs moteurs pouvant faire tourner ses roues afin d'effectuer des déplacements. Généralement, les roues du robot sont contrôlées par deux commandes motrices, soit une vitesse d'avancement et un taux de rotation. Habituellement, ces commandes s'expriment en mètres par seconde (m/s) et en degrés de rotation par Seconde (deg/s) [9].voir figure 1.5.

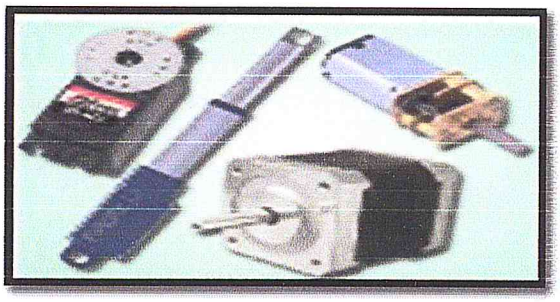


Figure 1.5 les différents actionneurs d'un robot.

1.8.2.4 Capteurs : Les capteurs ont pour fonction d'acquérir des données provenant de l'environnement. Les capteurs typiquement installés sur un robot mobile sont des sonars à ultrasons, un capteur laser de proximité, des encodeurs de roues (odomètres), une ou deux caméras optiques et des microphones. Les types d'informations perçues ainsi que leur précision varient beaucoup d'un capteur à l'autre [9].

1.8.2.5 Processeur : Le processeur est le cerveau du robot. Il calcule les mouvements des articulations du robot, détermine combien et à quelle vitesse chaque Joint doit se déplacer pour atteindre l'emplacement et la vitesse souhaitée, et supervise les actions coordonnées du contrôleur et les capteurs. Dans certains systèmes, le contrôleur et le processeur sont intégrés ensemble en une seule unité, et dans d'autres cas, ce sont des unités séparées [9]

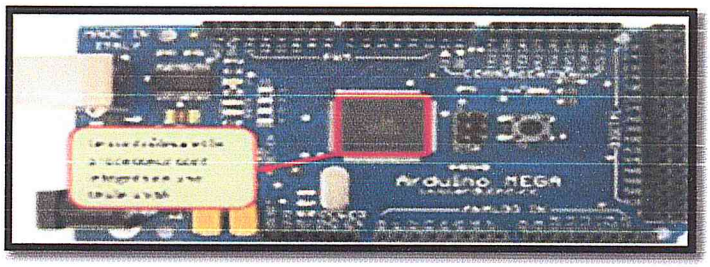


Figure 1.6 processeur utilisé pour commander un robot.

1.8.2.6 Logiciels : trois groupes de logiciels sont utilisés dans un robot. L'un est le système d'exploitation qui exploite le processeur.

Le second est le logiciel robotique qui calcule la motion nécessaire de chaque joint du robot basée sur des équations cinématiques. Ces informations sont envoyées au dispositif de commande. Ce logiciel peut être à différents niveaux, de la langue de la machine aux langues sophistiqués utilisés par les robots modernes. Les troisième groupes est la collection d'application - orientée les routines et les programmes développés pour utiliser le robot ou ses périphériques pour des tâches spécifiques telles que l'assemblage, le chargement de machines, la manutention et les routines de vision voir Figure1.7. [9].

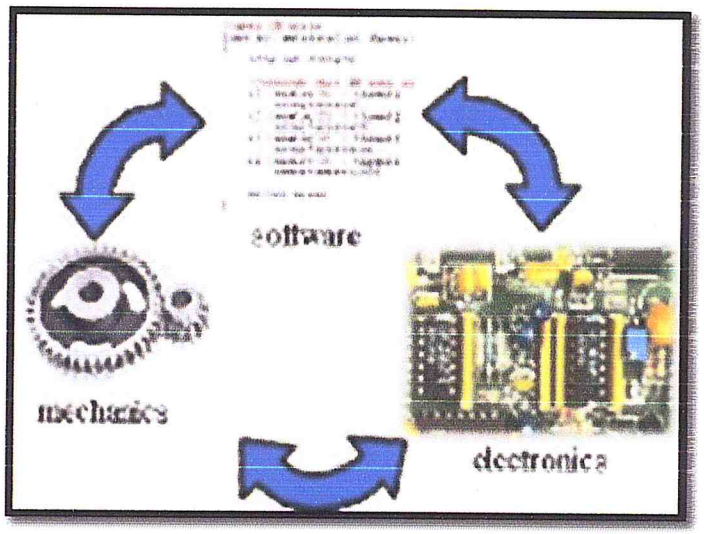


Figure 1.7 logiciels qui commandent un robot.

Afin de contrôler les robots et leurs introduire une certaine intelligence, les middlewares robotiques peuvent être d'une grande utilité.

1.9 Middleware robot :

Le middleware RT (Robot Technology Middleware) est une technologie qui implémente plusieurs concepts clés nécessaires au développement de systèmes de robots complexes, même dans des environnements géographiquement distribués. Grâce à une API utile, des composants standard et des canaux de communication réutilisables, et un certain degré

d'automatisation, le middleware RT aide l'utilisateur à créer des systèmes facilement reconfigurables. Le comportement des composants et leur mode d'interaction sont normalisés par la spécification RTC (Robot Technology Component). Jusqu'à présent, il existe deux implémentations de la spécification. OpenRTM.NET est écrit en .NET, tandis que l'autre implémentation, appelée OpenRTM-aist.

Dans la zone de la robotique, la tâche des systèmes de robots peut changer rapidement. Si le travail et les circonstances environnementales changent fréquemment, des composants réutilisables et reconfigurables sont nécessaires, en plus d'un cadre qui peut gérer ces changements. L'effort nécessaire pour développer de tels composants dépend du langage de programmation, de l'environnement de développement et d'autres cadres utilisés. Beaucoup de langages de programmation peuvent être appliqués pour ce processus, mais le développement d'un nouveau cadre est une tâche difficile. L'application d'un cadre existant en tant que système de base, tel que OpenRTM-aist dans le cas de ce chapitre, permet de simplifier considérablement les tâches de développement associées. Les Framework et les middlewares gagnent en popularité grâce à leur riche ensemble de fonctionnalités qui soutiennent le développement de systèmes complexes. Ensemble, tout cadre de robot et les pilotes de robot peuvent former un système complexe et efficace avec un effort relativement faible.

Dans de nombreux cas, la tâche du concepteur système est réduite à la configuration d'un cadre déjà existant. D'autre part, lorsqu'une structure existante ne requiert qu'un ensemble de fonctionnalités complètement nouvelles, elle peut simplement être étendue avec elles. L'amélioration d'un cadre existant est un travail plus facile que le développement d'un nouveau parce que la conception et la mise en œuvre d'un tel système nécessitent des connaissances et des compétences spécialisées (conception et mise en œuvre). Dans la plupart des cas, les fonctionnalités manquantes peuvent être intégrées dans le cadre existant, néanmoins il y a des cas où cela est difficile à réaliser. Un cadre existant peut être amélioré uniquement dans le cas où il est bien conçu et mis en œuvre. Malgré cela, la construction d'un tout nouveau système est un processus beaucoup plus long qui nécessite beaucoup d'efforts.

Dans le domaine de la robotique, il existe de nombreuses pièces de robots qui partagent des caractéristiques similaires, de sorte que le concept de middleware de robot comme cadre

commun pour les systèmes de robots complexes est évident. Probablement, il n'existe aucun cadre qui puisse satisfaire toutes les exigences ci-dessus [10].

1.9.1 Définition d'un middleware robot:

Middleware robotique est un middleware de logiciel qui étend la communication middleware tels que CORBA ou ICE. IL fournit Des outils, des bibliothèques, des API et des lignes directrices pour soutenir la création et le fonctionnement des deux composants du robot et des systèmes de Robots. Robot middleware agit également comme une colle qui établit une connexion entre les parties de robot de façon transparente [10].

1.9.2 Types de middleware robots :

Il y'en plusieurs types de robots middleware, on peut citer :Yarp, open-rtm, open – RDk et ros [10].Dans le cadre de notre projet nous allons utiliser le middleware ROS (Robot Operating System) car il a été proposé par notre établissement de stage le CDTA.

De plus, il doté d'une communauté actif et il permet de concevoir et programmer un Robot. Dans ce qui suit nous allons décrire le principe de ros.

1.9.3 Définition de ROS :

Comme son nom l'indique, ROS (Robot Operating System) est un système d'exploitation pour robots. De même que les systèmes d'exploitation pour PC, serveurs ou appareils autonomes, ROS est un système d'exploitation complet pour la robotique de service. ROS est un méta système d'exploitation, quelque chose entre le système d'exploitation et le middleware. Il fournit des services proches d'un système d'exploitation (abstraction du matériel, Gestion de la concurrence, des processus…) mais aussi des fonctionnalités de haut niveau (appels asynchrones, appels synchrones, base de données centralisée de données, système de paramétrage du robot…) [10].

1.9.4 Ajout d'un os pour un robot :

Avant l'arrivée des outils robotique, chaque concepteur de robot, chaque chercheur en robotique passait un temps non négligeable à concevoir matériellement son robot ainsi que le

logiciel embarqué associé (Notamment le soft qui le contrôle). Cela demandait des compétences dans plusieurs disciplines, notamment en mécanique, électronique et programmation embarquée. Généralement, les programmes ainsi conçus correspondaient plus à de la programmation embarquée (programmation sur les plateformes robotique), proche de l'électronique, qu'à de la robotique proprement dite, telle que nous pouvons la rencontrer aujourd'hui dans la robotique de service. La réutilisation des programmes était non triviale, car ils Étaient fortement liés au matériel sous-jacent.

Comme la robotique nécessite des compétences très différentes, qui sont généralement hors de la portée d'une seule et même personne (d'où la notion d'équipe dans la Recherche), ROS présente un autre avantage, celui de rassembler des savoir-faire de différentes disciplines [10]. En effet, concevoir et programmer un robot, c'est :

- Gérer le matériel en écrivant les pilotes.
- Gérer la mémoire et les processus.
- Gérer la concurrence et la fusion de données.
- Proposer des algorithmes de raisonnement abstrait faisant largement appel à L'intelligence artificielle

1.9.5 Format générale de ROS :

La philosophie de conception de ROS se résume dans les principes suivants :

a) **Peer to Peer :** Un robot suffisamment complexe est composé de plusieurs ordinateurs ou cartes embarquées reliées par Ethernet ainsi que parfois des ordinateurs externes au robot pour des tâches de calcul intensif.

Une architecture peer to peer couplée à un système de tampon (buffering) et un système de lookup (un name service appelé Master dans ROS), permet à chacun des acteurs de dialoguer en direct avec un autre acteur, de manière synchrone ou asynchrone en fonction des besoins [10].

b) **Multi langages:**

ROS est neutre d'un point de vue langage et peut être programmé en différents langages. La spécification de ROS intervient au niveau message. Les connexions peer to peer sont négociées en XML-RPC qui existe dans un grand nombre de Langages. Pour supporter un nouveau langage, soit on rewrappe les classes C++ (ce qui a été fait pour le client Octave par exemple), soit on écrit les classes permettant de générer les messages. Ces messages sont décrits en IDL (Interface Definition Language) [10].

***c)* Basé sur des outils:**

Plutôt qu'un runtime monolithique, ROS a adopté un design microkernel qui utilise un grand nombre de petits outils pour faire le build et le run des différents composants ROS. Lorsque vous parcourrez les tutoriaux ROS, vous apprenez à vous servir de plusieurs commandes permettant de manipuler les nœuds et les messages. Chaque commande est en fait un exécutable. L'avantage de cette solution est qu'un problème sur un executable N'affecte pas les autres, rendant le système plus robuste et évolutif qu'un système basé sur un runtime centralisé [10].

***d)* Léger :**

Afin de lutter contre le développement d'algorithmes plus ou moins liés avec l'OS robotique et donc difficilement réutilisables ensuite, les développeurs de ROS souhaitent que les pilotes et autres algorithmes soient contenus dans des exécutables indépendants. Cela assure la réutilisabilité maximale et surtout le maintien d'une taille réduite. Ce mécanisme rend ROS facile d'usage, la complexité se trouvant dans les librairies [10].

***e)* Gratuit ET Open Source:**

Nous avons déjà expliqué les raisons de ce choix avant. Notons toutefois que l'architecture choisie est cohérente avec ce choix: ROS passe des données grâce à de la communication interprocessus. De ce fait, les modules n'ont pas besoin d'être liés dans un unique processus, facilitant ainsi l'usage des différentes licences [10].

1.10 Systèmes de fichiers ROS :

Les ressources de ROS sont organisées dans une structure hiérarchique sur disque. Deux concepts importants se détachent :

- Le package :

C'est l'unité principale d'organisation logicielle de ROS. Un package est un répertoire qui contient les nœuds (nous verrons ci-dessous ce qu'est un nœud), les librairies externes, des données, des fichiers de configuration et un fichier de configuration xml nommé manifest.xml [10].

- La stack :

Une stack est une collection de packages. Elle propose une agrégation de Fonctionnalités telles que la navigation, la localisation… Une stack est un répertoire qui contient les répertoires des packages ainsi qu'un fichier de configuration nommé stack.xml. En plus de ces deux notions très importantes, on relève également la notion de distribution. Une Distribution, comme dans Linux, est un ensemble de stacks visionnées [10].

1.11 Architecture de ROS :

Le principe de base d'un OS robotique est de faire fonctionner en parallèle un grand nombre d'exécutables qui doivent pouvoir échanger de l'information de manière synchrone ou asynchrone. Par exemple, un OS robotique doit interroger à une fréquence définie les capteurs du robot (capteur de distance à ultrasons ou infrarouge, capteur de pression, capteur de température, gyroscope, accéléromètre, Caméras, Microphones…), récupérer ces informations, les traiter (faire ce que l'on appelle-la fusion de données), les passer à des algorithmes de traitement (traitement de la parole, vision artificielle, localisation et cartographie simultanée,…) et enfin contrôler les moteurs en retour. Tout ce processus s'effectue en continue et en parallèle. D'autre part, L'OS robotique doit assurer la gestion de la concurrence afin d'assurer l'accès efficace aux ressources du robot. Nous décrivons ci-dessous les concepts regroupés dans ROS sous le nom de « ROS Computation Graph » et qui permettent d'atteindre ces objectifs. Il s'agit des concepts utilisés par le système en cours de fonctionnement tandis que le «ROS FileSystem » décrit dans le paragraphe précédent correspond aux concepts statiques [10].

1.11.1 Les Nœud :

ROS répond à tout cette problématique grâce à des notions de base simples. La première notion est la notion de nœud. Dans ROS, un nœud est une instance d'un exécutable. Un

nœud peut correspondre à un capteur, un moteur, un algorithme de traitement, de surveillance… Chaque nœud qui se lance se déclare au Master. On retrouve ici l'architecture microkernel où chaque ressource est un nœud indépendant [10].

1.11.2 Le Master :

Le Master est un service de déclaration et d'enregistrement des nœuds qui permettent ainsi à des nœuds de se connaitre et d'échanger de l'information. Le Master est implémenté via XMLRPC. Le Master comprend une sous-partie très utilisée qui est le paramètre Server. Celui-ci, également implémenté sous forme de XMLRPC, comme son nom l'indique est une sorte de base de données centralisée dans laquelle les nœuds peuvent stocker de l'information et ainsi partager des paramètres globaux [10].

1.11.3 Les Topics :

L'échange de l'information s'effectue 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). Un ou plusieurs nœuds pourront 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 un peu comme un Flux RSS. 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 [10].

1.11.4 Les Messages :

Un message est une structure de donnée composite. Un message est composé d'une combinaison de types primitifs (chaines de caractères, 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 (selon ce que vous aurez programmé) avec un message contenant par exemple un entier représentant la position du moteur, un flottant représentant sa température, un autre flottant représentant sa vitesse a description des messages est stockée dans nom_package/msg/monMessageType.msg Ce fichier décrit la structure des messages [10].

1.11.5 Les Services :

Le topic 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 procedure call). La description des services est stockée dans nom_package/srv/monServiceType.srv. Ce fichier décrit les structures de données des requêtes et des réponses [10].

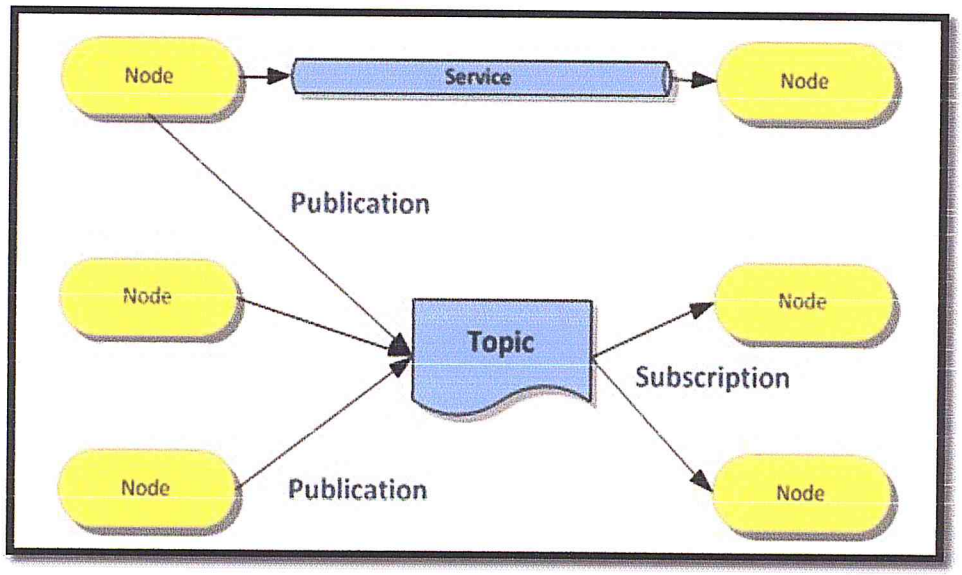


Figure 1.8. Fonctionnement du système ROS.

1.12 Conclusion :

Pour conclure, la robotique est un domaine complexe du fait que son évolution et liés aux actions et aux modèle de notre vie quotidienne qui se change et s'évolue de jour à jour ce qu'on a constaté en réalisant ce chapitre. Dans ce qui suit nous allons faire explorer les différents planificateurs pour un robot manipulateur mobiles.

Chapitre 2

Planificateurs de taches pour robots manipulateur mobile

2.1 Introduction :

Le problème de planification en intelligence artificielle consiste à déterminer quelles sont les actions à exécuter, et ce, dans quel ordre, pour accomplir un but donné. Dans ce chapitre, nous introduisons les techniques de planification les plus répandues. Nous allons nous intéresser plus précisément à la planification en robotique. Nous allons commencer dans ce chapitre par définir ce qu'est la planification, puis nous décrirons les méthodes de planification en général. Nous présenterons ensuite .

2.2 La planification :

La planification en Intelligence Artificielle concerne la résolution des problèmes dans plusieurs domaines [Ghallab et al. 04]. Les solutions de ces problèmes sont sous la forme d'une suite d'opérations ou d'actions à réaliser dans le but d'atteindre une l'objectif. Une telle solution est appelée un plan. Le programme qui est capable de trouver un plan est un planificateur. Comme illustré à la figure 2.1, un planificateur reçoit généralement en entrée un domaine, le modèle des actions et du monde (environnement), un état initial représentant l'état actuel du système et un but à atteindre. En sortie, le planificateur génère un plan. Ce plan peut prendre différentes formes, la plus commune étant une séquence d'actions ou de tâches primitives. Intuitivement, une action ou une tâche primitive est une activité que le robot sait déjà comment accomplir de manière innée, par l'agencement d'un ou plusieurs modules comportementaux.

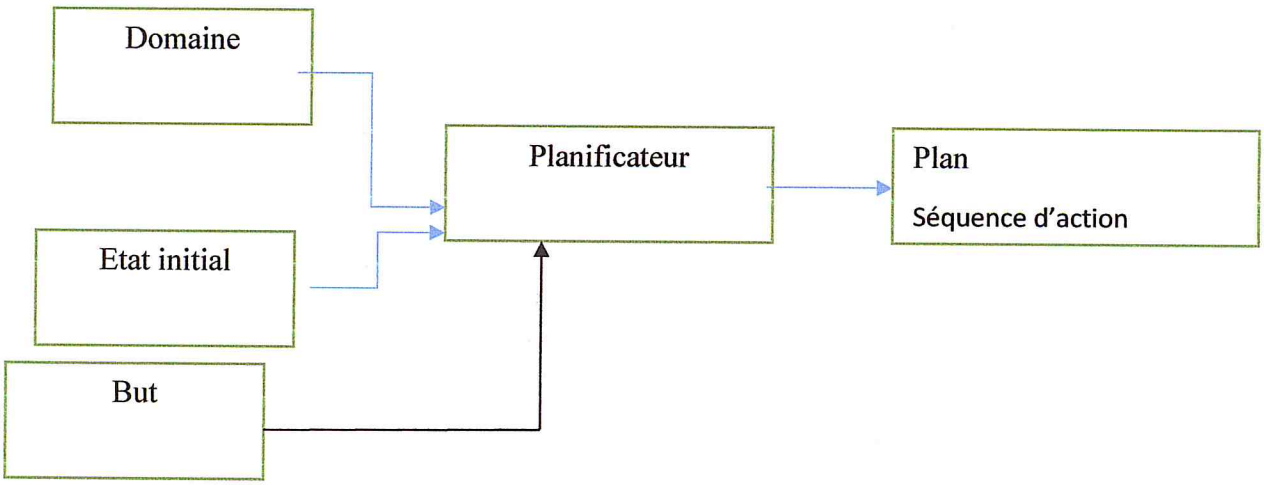


Figure 2.1 un modèle de planification.

2.3 Les outils de représentation de problèmes de planification de tâches :

Les méthodes d'ordonnancement les plus connues sont :

- Le diagramme de Gantt
- La méthode MPM
- Le PERT

Dans ce qui suit nous allons résumer les trois méthodes.

2.3.1 Diagramme de Gantt :

Le diagramme de Gantt est un outil permettant de modéliser la planification de tâches nécessaires à la réalisation d'un projet. Le principe de ce type de diagramme est de représenter au sein d'un tableau, en ligne les déférentes tâches et en colonne les unités de temps (exprimées en mois,semaines, jours,...) [11]. Les déférentes étapes de réalisation d'un diagramme de Gantt sont les suivantes :

- Définition des déférentes tâches à réaliser et leurs durées.
- Définition des relations d'antériorité entre tâches.
- Représentation des tâches par des traits dans le diagramme : d'abord les tâches n'ayant aucune antériorité, puis celles dont les tâches antérieures ont déjà été représentées, et ainsi de suite...
- Représentation de la progression réelle du travail par un trait pointillé parallèle à la tâche planifiée.

La figure 2.2 représente un diagramme de Gantt où chaque colonne représente une unité de temps, les traits épais représentent les durées d'exécution prévues des tâches et les traits pointillés représentent le déroulement d'exécution. Par exemple, la tâche B, qui dure 5 unités de temps, ne peut commencer son exécution qu'après la _n de la tâche A et elle peut s'exécuter en même temps que la tâche C.

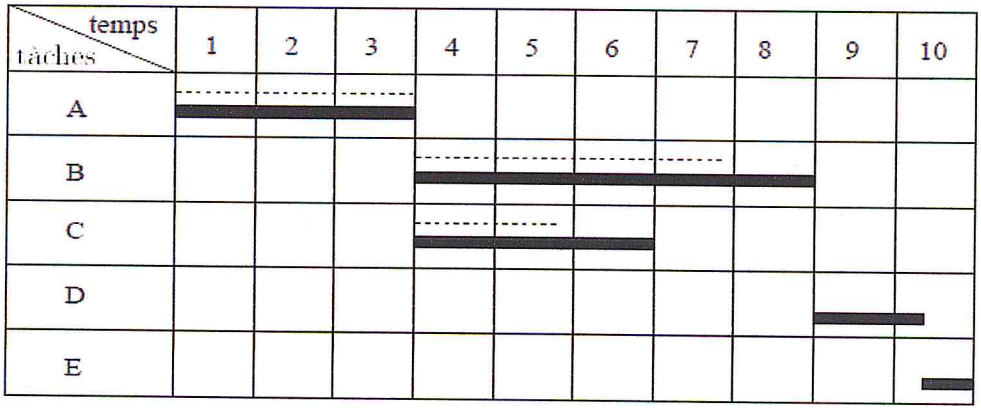


Figure 2.2 un exemple de diagramme de Gantt

2.3.2 La méthode MPM :

Dans la méthode des potentiels-métra, le problème est représenté sous forme d'un graphe tel que les tâches sont représentées par des nœuds et les contraintes de succession par des arcs. À chaque nœud sont associées une date de début au plus tôt et une date de _n au plus tard. À chaque arc est associé un délai d'attente entre les tâches. La figure 2.3 représente un exemple de la méthode des potentiels-métra. La date de début au plus tôt d'une tâche dépend de la date de _n des tâches qui la précèdent. La tâche DEBUT est initialisée avec une date de début au plus tôt égale à zéro. Cette méthode permet de déterminer la date de réalisation d'un projet ainsi que la date de début et de _n de chaque tâche mais elle est incapable de résoudre des problèmes qui prennent en compte plus de contraintes telles que l'incertitude et les coûts d'exécution des tâches.

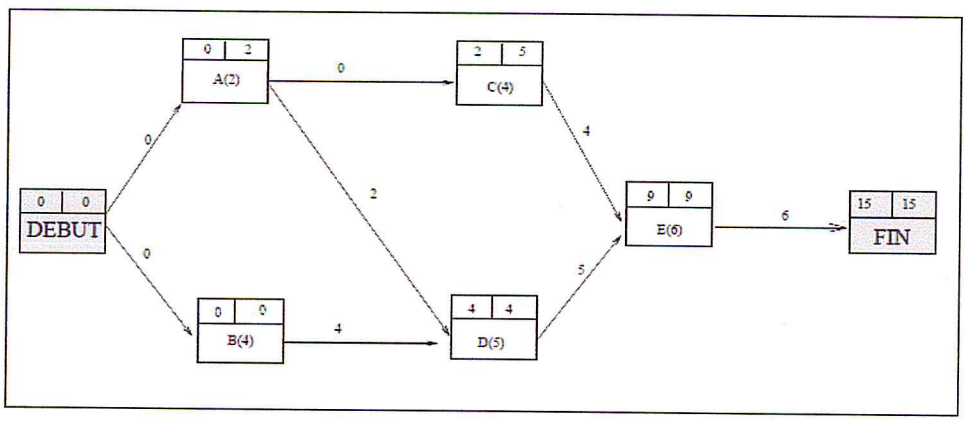


Figure 2.3 un exemple d'un graphe MPM

2.3.3 Le PERT :

La méthode PERT consiste à mettre en ordre sous forme de réseau plusieurs tâches qui grâce à leurs dépendances et à leur chronologie permettent d'avoir un produit fini. Elle représente le problème sous forme d'un graphe tel que les tâches sont représentées par un arc auquel on associe un nombre entre parenthèses qui représente la durée de la tâche. Un nœud représente la fin d'une ou de plusieurs tâches. La figure 2.4 représente un graphe PERT.

La construction d'un tel graphe se fait par niveau : le niveau 1 contient les tâches sans antécédents, le niveau i contient les tâches dont les antécédents sont exclusivement du niveau i−1. La date de début au plus tôt d'une tâche dépend des dates de fin des tâches qui la précèdent. Cette méthode permet de déterminer la date de début et de fin de chaque tâche ainsi que le chemin critique c'est-à-dire un ensemble d'activités tel que tout retard dans leur exécution provoquerait un retard de la fin du projet. Par contre, elle ne présente pas d'échelle calendaire comme dans le cas de diagramme de Gantt.

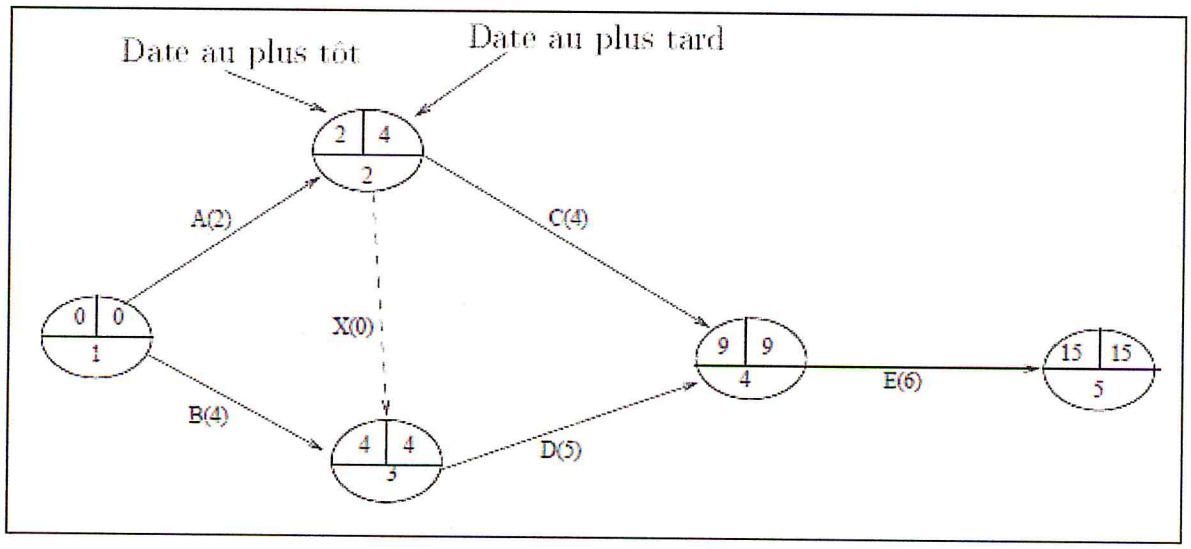


Figure 2.4 un exemple de diagramme de pert.

2.4 Planification classique :

Dans la littérature, la planification classique est souvent associée à STRIPS (Standford Research Institue Problem Solver) [12], un des premiers planificateurs à être développés (1972).

2.4.1 STRIPS :

STRIPS apparu en 1970, est un des premiers planificateurs dans le domaine de l'Intelligence Artificielle. Il définit les actions sous forme d'opérateurs, chacun est composé des pré-conditions (Propriétés) nécessaires à l'exécution de l'action et des effets engendrés par sa mise en application. Les pré-conditions et les effets forment une proposition qui peut être vraie ou fausse. Un effet, sous la forme ajouter à la liste ou supprimer de la liste, a comme objectif de rendre la proposition vraie. La figure 2.5 représente un opérateur STRIPS composé de ses préconditions, de ses effets et de son action. Considérons par exemple, un robot qui va ouvrir une porte. Porte ouverte et porte fermée sont deux propositions telles que, à un moment donné, l'une est vraie et l'autre est faux. L'action ouvrir la porte doit avoir la proposition porte fermée comme une pré-Condition et comme un effet supprimer de la liste et la proposition porte ouverte comme un effet ajouter à la liste. En outre la figure 2.6 illustre l'opérateur ouvrir. La terminologie de STRIPS est très intéressante dans la mesure où elle est simple et permet de modéliser beaucoup de problèmes de planification, ce qui prouve son utilisation dans plusieurs formalismes en particulier le langage ADL6 [Pednault 89] en ajoutant des pré-conditions négatives, des effets conditionnels, des variables, etc. Par contre, dans ce planificateur, démontré PSPACE-complet, toutes les actions sont déterministes; c'est à dire que tous les effets de l'action sont connus et bien déterminés et qu'une action ne peut se déclencher que dans un certain contexte. L'algorithme général de STRIPS est représenté dans l'algorithme 1 et est du type suivant : le démonstrateur cherche à prouver que l'union de l'état initial E0 et de la négation du but B0 est inconsistante (donc B0 est vérifié dans l'état E0). Depuis ce système fondateur, et dans le cadre « classique »de la planification qu'il a délimité (environnement statique, observabilité totale, agent omniscient, actions Atomiques et déterministes), de très gros progrès ont été réalisés.

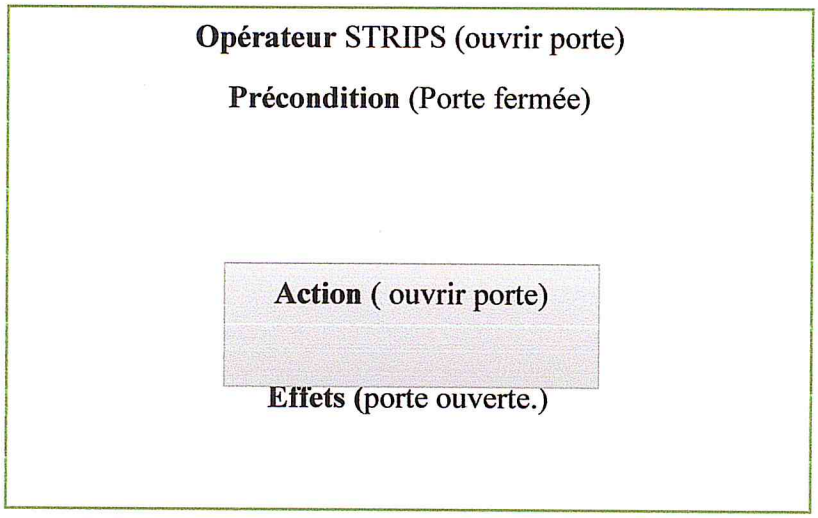


Figure 2.5 Exemple d'un opérateur STRIPS

2.5 Algorithmes de planification déterministe :

Ils existent plusieurs techniques de planification déterministe, citant entre

2.5.1 Recherche à chainage avant dans un espace d'état :

Une technique simple pour résoudre un problème de planification déterministe est l'utilisation d'un algorithme de recherche à chaînage avant dans l'espace des états du domaine. La figure 2.6 illustre un exemple d'une telle recherche dans le domaine de la livraison de colis. En partant de l'état initial, il suffit d'appliquer toutes les actions applicables. Chaque action est appliquée et mène à un nouvel état. De façon récursive, cette opération est répétée à partir des nouveaux états trouvés, et ce, jusqu'à ce qu'un état satisfaisant au but donné soit atteint. Dans le cas où aucune solution n'existe, alors tout l'espace d'états atteignable est exploré De façon exhaustive. Pour parcourir l'espace d'états, l'algorithme A* [13] (Recherche heuristique) peut être utilisé. Une fonction heuristique souvent utilisée, estimant le coût restant pour atteindre le but g à partir de l'état s'est la fonction suivante : $h(s) = |g - s|$. Cette fonction compte le nombre de littéraux restants à obtenir pour atteindre le but donné. Bien que cette fonction soit admissible, elle n'est pas très efficace puisque sa valeur est généralement beaucoup plus petite que le coût restant. Pour A*, une fonction heuristique est dite admissible si elle ne surestime pas le coût restant. Avec une telle fonction, A* trouve une solution optimale. Parmi

les fonctions heuristiques admissibles, celle qui sous-estime le moins le coût restant est la plus efficace puisqu'elle permet d'obtenir plus rapidement une solution en explorant un plus petit espace d'états. Par Exemple, dans l'exemple de la figure 2.6, tous les états ont une valeur d'heuristique égale à un (1) sauf l'état final où la valeur est de zéro (0). Pour cette raison, sur cet exemple-ci, l'algorithme A* se comporte comme une recherche en largeur puisque l'heuristique ne permet pas de trier efficacement les états en fonction de leur proximité avec le but. Cependant, il existe d'autres fonctions heuristiques plus efficaces.

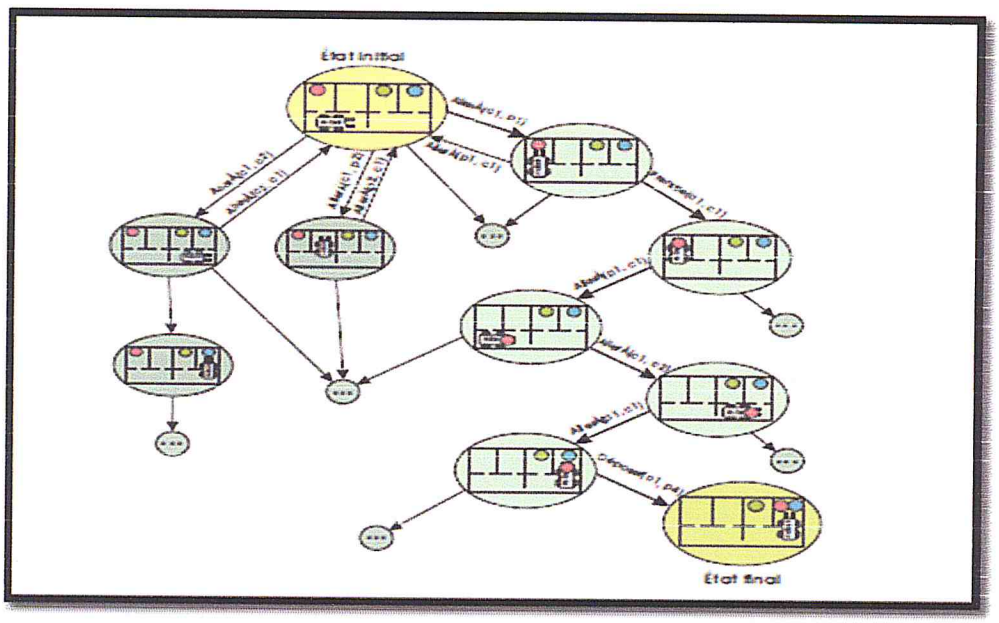


Figure 2.6 recherche à chainage avant.

Sur le même principe que la recherche vers l'avant, on peut également effectuer une recherche à chaînage arrière dans l'espace d'états. Au lieu de partir de l'état initial, la recherche débute du but et cherche toutes les actions permettant d'atteindre ce but. Pour chaque action possible, un état prédécesseur est trouvé. Ce processus se répète jusqu'à ce que l'état initial du problème donné soit trouvé. Théoriquement, une recherche vers l'arrière a la même efficacité qu'une recherche vers l'avant. Par contre, il peut arriver en pratique que sur certains domaines l'une des deux recherches soit meilleure que l'autre.

2.5.2 Recherche dans l'espace de plan :

Une autre approche plus élaborée est de faire une recherche dans l'espace de plans. Les nœuds de cet espace sont des spécifications partielles de plans et les arcs sont des opérations de raffinement pour résoudre un but n'étant pas encore satisfait ou pour résoudre une contrainte. Une solution obtenue par ce type de recherche n'est plus un plan sous forme de séquence d'actions, mais un plan partiellement ordonné noté π= (A, $\prec$, B, L) où A = {a1, a2, ..., an} est un ensemble d'actions et $\prec$ est un ensemble d'ordres partiels de la forme ai $\prec$ aj, indiquant que l'action ai doit précéder l'action aj Pour plus de détails, le lecteur est référé à [14].

2.5.3 Recherche dans l'espace d'un graphe de planification :

Une technique encore plus sophistiquée est l'approche GraphPlan [15]. Elle consiste à faire une recherche dans un graphe de planification. La construction du graphe de planification se fait niveau par niveau, où chaque niveau ni contient l'ensemble des prédicats accessibles après l'application d'au moins i actions. Le premier niveau, soit n0, contient un nœud pour chaque prédicat de l'état initial. Un niveau ni+1 est formé par l'union des prédicats du niveau ni et de tous les prédicats obtenus par l'application de toutes les actions applicables au niveau ni. Entre deux niveaux consécutifs ni et ni+1, des arêtes étiquetées d'une action indiquent omment les prédicats d'un niveau ni+1 dépendent des prédicats du niveau précédent ni. Aussi, à l'intérieur d'un même niveau, d'autres arêtes indiquent les prédicats étant mutuellement exclusifs. Enfin, une fois le graphe de planification construit jusqu'à un certain niveau, GraphPlan fait une recherche dans ce dernier pour trouver un plan.

2.6 Graph plan :

GraphPlan est l'un des planificateurs classiques les plus connus grâce à ses performances le distinguant par rapport à tous ses prédécesseurs. Son apparition a donné une évolution importante aux recherches dans le domaine de la planification en IA.

Celles-ci ont permis aux algorithmes de planification STRIPS d'augmenter leurs performances de manière si importante que l'on peut maintenant envisager des applications réelles à moyen terme. Il s'agit d'un générateur de plans complets, qui utilise une description du problème basée sur le

formalisme STRIPS, et autorise une certaine parallélisation des actions. La contribution la plus importante de GraphPlan est une structure compacte appelée Graphe de planification, qui entrelace des niveaux de nœuds de propositions et d'actions construits en chaînage avant, plus des exclusions mutuelles qui représentent l'impossibilité pour deux actions d'être exécutées en parallèle, ou pour deux propositions d'être présentes dans un même état après l'application d'un plan parallèle d'une longueur donnée. Cette structure est ensuite explorée par un algorithme de recherche en chaînage arrière. Le graphe de planification est ensuite étendu niveau par niveau jusqu'à ce qu'une solution soit trouvée ou qu'une condition d'arrêt soit vérifiée. Le graphe de planification peut aussi être traduit-en une formule booléenne dont les modèles correspondent aux plans solutions. La taille et le temps de création de ce graphe sont polynomiaux par rapport à la taille du problème. Un plan en GraphPlan est une séquence d'ensembles d'actions. Chacun de ces ensembles contient des actions qui doivent être exécutées simultanément selon l'ordre de l'ensemble dans le plan.

2.7 Planification hiérarchique :

Une autre approche pour spécifier des connaissances stratégiques est d'utiliser des réseaux hiérarchiques de tâches, ou en anglais, Hierarchical Task Network (HTN) [16]. La planification HTN est basée sur la notion de tâches. Au lieu de spécifier un but sous la forme d'une conjonction de clauses, un but est défini sous la forme d'une liste de tâches de haut niveau. Un planificateur HTN fait alors une recherche dans un espace de décomposition de tâches afin de réduire la mission en une liste de tâches primitives. Pour chaque tâche de haut niveau, un modèle de décomposition dicte comment réduire cette tâche en sous-tâches .Il peut y avoir plusieurs façons de décomposer une tâche. Ces connaissances sur les différentes décompositions possibles relèvent de stratégies habituellement possédées par un expert. Plus formellement, un domaine de planification HTN est défini par le quadruplet D = (_, M, O, N) où _, M, et O sont, tout comme pour la planification classique, respectivement l'alphabet des symboles du domaine, le monde et l'ensemble des opérateurs. Enfin, N est la définition du réseau de décomposition de tâches, soit un ensemble de méthodes de décomposition. Une méthode de décomposition (sous-réseau) $n \in N$ est défini comme suit : n = (tache(n), precond(n), Soustaches(n), $\prec$) où tache(n) est une tâche de haut niveau, precond(n) est un ensemble de préconditions, soustaches(n) est l'ensemble des

sous-tâches et $\prec$ est un ensemble d'ordres partiels sur l'ensemble des sous-tâches. Une sous-tâche peut être une tâche primitive ou une autre tâche de haut niveau. La figure 2.7 montre un exemple de réseau de décomposition de tâches pour le domaine de la livraison de colis. Dans le haut de la figure, il y a une méthode de décomposition pour la tâche de haut niveau Livrer Objet (o, orig, Dest) qui consiste à livrer l'objet o d'à l'endroit orig à l'endroit Dest. Elle se décompose en les quatre Sous-tâches suivantes : SeRendreÀ (orig), Prendre (o, orig), SeRendreÀdest et Déposer (o, Dest). La flèche horizontale, qui traverse les quatre flèches liant la tâche de haut niveau aux tâches de niveau inférieur, indique un ordre chronologique entre les sous-tâches. Enfin, les deux autres méthodes de décomposition décrivent comment une tâche SeRendreÀ peut être décomposée en une suite de tâches primitives AllerÀ donnant un chemin d'endroits djacents. La famille des planificateurs SHOP (Simple Hierarchical Ordered Planner), soit JSHOP, SHOP [17] et SHOP2 [17] sont des exemples de planificateurs basés sur la planification HTN.

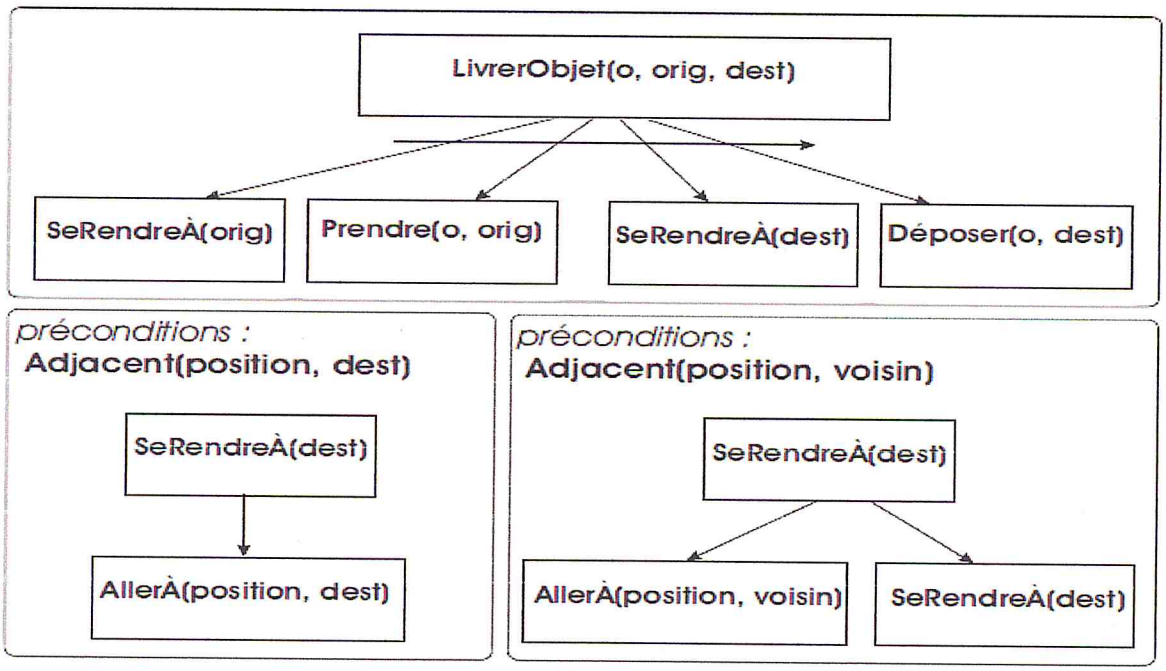


Figure 2.7 exemple de planification en réseau hiérarchique.

2.8 La planification heuristique :

Il est possible d'utiliser comme fonction heuristique le compte de prédicats manquants dans un état pour atteindre le but. Une meilleure approche consiste à utiliser une version allégée des graphes de planification (relaxed Graph-Plan) afin d'estimer le nombre d'actions requises pour atteindre le but. En d'autres mots, au cours de la recherche, pour chaque état visité, un graphe de planification simplifié est construit temporairement, mais sans prendre en compte les prédicats mutuellement exclusifs et les effets négatifs. Lorsqu'un niveau contient tous les prédicats du but recherché, le nombre d'actions minimales requises pour générer ces prédicats est compté. Puisque les prédicats mutuellement exclusifs et les effets négatifs sont ignorés, cette fonction heuristique ne surestime jamais le coût restant ; elle est donc admissible. Cette heuristique produit des valeurs plus près du coût restant que l'heuristique classique comptant les prédicats manquants dans l'état. Elle est donc plus efficace et permet de réduire considérablement le nombre d'états visités au cours de la recherche. Grâce à cette technique, le planificateur Fast-Forward (FF) [18] a obtenu la meilleure performance dans la catégorie planification classique (STRIPS) 4 à la 3e Compétition IPC (International Planning Competition) lors de la conférence ICAPS (International Conference on Autmated Planningand Scheduling) en 2002.

2.9 Planification concurrente :

Parfois, il peut être nécessaire de générer des plans dans lesquels plusieurs actions peuvent s'exécuter simultanément. Il s'agit de planification concurrente. Pour ce faire, l'hypothèse H4 doit être levée et il faut créer un nouveau modèle de planification permettant de gérer des actions avec des durées variables. Une technique pour résoudre ce type de problème consiste à ajouter une variable temporelle directement dans la représentation interne des etats et d'ajouter une durée dans la description des opérateurs [19]. De plus, les prés conditions et les effets sont temporels : on peut les spécifier pour le début, pour la fin ou pour toute la durée de l'action. Le planificateur SAPA [20] utilise cette approche afin de planifier des actions simultanées

2.10 Planification avec métrique:

Pour beaucoup de domaines, certaines caractéristiques de l'environnement, comme des ressources, doivent être modélisées à l'aide de variables numériques. Par exemple, le niveau des batteries d'un robot peut se représenter à l'aide d'une variable réelle. Le temps peut aussi être considéré comme une ressource particulière ne pouvant qu'être incrémentée. La spécification des opérateurs peut définir des préconditions sur ces variables numériques ainsi que des effets pouvant les incrémenter ou les décrémenter. Par exemple, un opérateur de déplacement pour un robot mobile peut avoir la précondition d'avoir suffisamment d'énergie et l'effet de décrémenter la variable du niveau d'énergie, tandis qu'un opérateur de recharge peut avoir comme effet d'incrémenter cette même variable. En planification classique, l'efficacité d'un plan se mesure en termes de nombre d'actions qu'il contient ; un plan est optimal s'il n'existe pas d'autres plans avec moins d'actions. En d'autres mots, les actions ont tout le même coût. La planification avec métriques va plus loin : elle permet d'avoir des actions avec des coûts variables. Deux approches équivalentes peuvent être utilisées pour estimer le coût d'un plan :

Additionner les coûts individuels des actions, ou évaluer une expression mathématique basée sur les valeurs numériques de l'état final. On peut par exemple demander à un planificateur de trouver un plan minimisant un certain critère comme l'énergie consommée, la durée totale ou même une combinaison linéaire de ces deux critères. Intégrer la gestion de métriques dans un planificateur à chaînage avant est plutôt simple : Il ne s'agit que d'ajouter des variables numériques dans la représentation des états. Par contre, pour être efficace, le planificateur doit tenir compte des ressources lors de son raisonnement. Une technique efficace consiste à étendre le concept de la planification heuristique basée sur les graphes de planification en considérant les effets numériques comme suit : les incrémentations sont considérées comme des effets positifs et les décrémentations sont ignorées comme ce fut le cas pour les effets négatifs. Basé sur cette approche, le planificateur Metric-FF [21], qui est une extension métrique à FF [22], a terminé en première position pour les problèmes de type numérique à la 3e compétition IPC. Les planificateurs LPG [23] et SAPA [24] exploitent aussi cette même approche.

2.11 Planification non déterministe :

La planification non déterministe consiste à générer des plans conditionnels prévoyants des alternatives pour des cas non déterministes, C'est-à-dire connus à l'avance, mais avec un certain degré d'incertitude. La planification classique fait l'hypothèse que le domaine est complètement déterministe. En d'autres mots, on présume que les plans s'exécuteront dans un monde idéal où l'échec n'existe pas. Par contre, la réalité est tout autre et des situations imprévues peuvent survenir. Par exemple, dans le domaine de la livraison de colis, il est possible que le robot puisse se tromper de pièce suite à l'exécution d'une action de déplacement. Ainsi, le résultat de l'exécution d'une action n'a plus qu'un seul état successeur, mais un ensemble d'états successeurs possibles où chacun peut avoir une certaine probabilité. Ce type de planification est d'une complexité de plusieurs ordres de grandeur supérieurs à la planification déterministe. Comme notre approche n'est pas basée cette méthode, ce type de planification n'est pas exploré dans ce mémoire. En fait, nous expliquons comment, avec des techniques simples [25], il est possible d'utiliser un planificateur déterministe pour générer et exécuter des plans dans des environnements dynamiques et non déterministes. En d'autres mots, au lieu de faire face au non-déterminisme durant la planification, le robot l'ignore et planifie pour le meilleur des mondes. Cependant, le robot surveille le déroulement de l'exécution pour détecter les situations divergentes du monde parfait. Dans ce cas, le planificateur corrige le plan et le robot continu à nouveau.

2.12 Planification et exécution :

Le contrôle d'exécution de plans est un problème Particulièrement difficile lorsqu'il doit être effectué à bord de systèmes autonomes tels que des robots.

Un tel système doit disposer de processus de délibération pour engendrer des plans qui réalisent les buts de la mission tout en respectant des délais et des contraintes de précédence.

La figure 2.8 représente un exemple d'un système de planification qui intègre un contrôleur d'exécution.

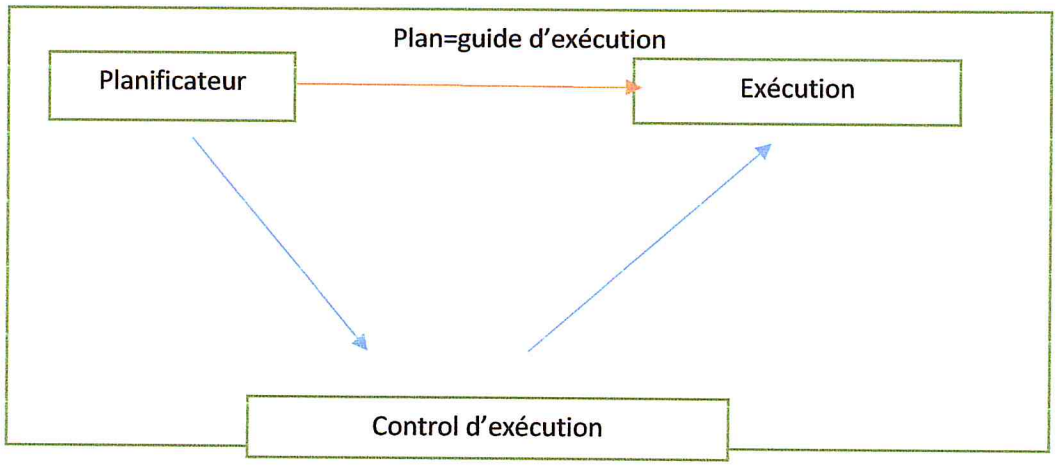


Figure2.8 exemple d'un system de planification simple

pour remédier aux échecs lors de l'exécution du plan (dûs à des événements imprévus, au manque de précision dans la description des opérateurs ou du monde,...). Nous citons brièvement les principales techniques utilisées en cas d'échec :

1. Re-planifier lorsqu'une différence entre l'état du monde attendu et l'état constaté Apparaît.
2. Lier différemment les processus de génération et d'exécution des plans. Ceci rend Le planificateur plus sensible aux interactions, mais plus réactif aux Changements imprévus de l'univers [4].
3. Anticiper les échecs et planifier pour les résoudre avant qu'ils ne se présentent [4].
4. Lier la planification, l'exécution d'actions et la surveillance de l'exécution dans un Même système.
5. Contrôler la situation en lançant un système d'exécution indépendant dès qu'il est possible d'agir sans l'aide d'un plan. Ce dernier sert simplement à augmenter les capacités du système à atteindre les buts. Certaines erreurs lors de l'exécution paraissent inévitables : elles sont généralement dues à des événements imprévus du monde réel sur lesquels on ne peut avoir aucun contrôle (figure 2.9).

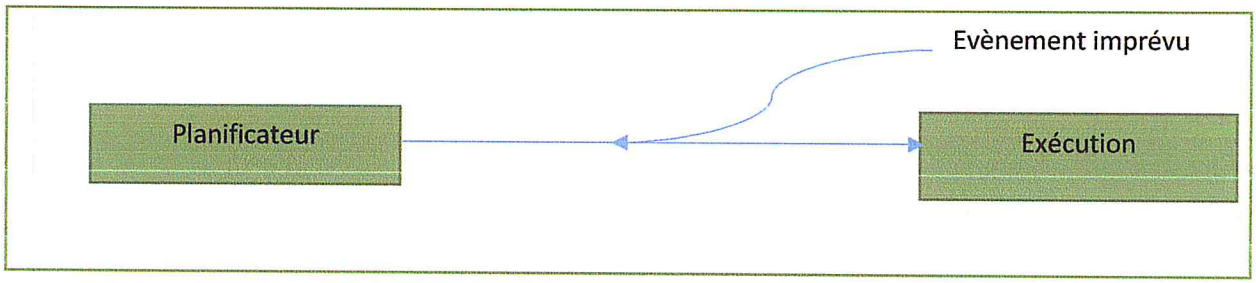


Figure2.9 Événements imprévus entre la planification d'une action et son exécution.

2.13 Conclusion :

Dans cette section nous avons effectué une étude sur les planificateurs de tâches dédiés à différents types de problèmes pratiques. Notre choix s'est porté dans ce mémoire sur l'utilisation des planificateurs de type HTN. Ces derniers sont fortement adaptés à notre problème. Dans ce qui suit nous allons présenter les bases de notre travail avec l'application d'une planification hiérarchique afin de planifier les tâches exécutées par le robot au niveau du CDTA.

Chapitre 3
Conception

Introduction :

Dans le chapitre précédent on a fait une étude sur les différents planificateurs de tâches quel soit dédiée au domaine générale ou qui sont utilisé dans le domaine de la robotique. Après cette étude, on choisit la planification hiérarchique dans la réalisation de notre travail.

3.1 Problématique :

- Afin que les robots puissent coopèrer avec l'être humain et peuvent réaliser les taches dédiées à l'homme, il faut qu'ils aient un comportement intelligent .Ce processus de raisonnement est nommé la planification des taches.
- La planification des taches en intelligence artificielle consiste à déterminer quelles sont les taches a exécutées et dans quelle ordre. De plus, elle est considérée comme une étape primordiale et difficile dans la réalisation des robots vu le degré de complexité qui est lié aux tâches complexes pouvant être résolut par le robot manipulateur mobile.
- L'objectif de notre projet est d'appliquer la méthode hiérarchique un robot manipulateur mobile afin qu'il puisse exécuter les taches d'une manière intelligente. La planification hiérarchique permet de définir des niveaux hiérarchiques afin de décomposer une tâche complexe en des sous-taches .Ce niveaux sont définis comme suit :
- Une couche décisionnelle dans laquelle une tâche complexe à exécuter est définie. Dans cette partie l'objectif à atteindre par le robot (suivre une cible, saisir un objet, naviguer …) est décrit.
- Dans une seconde couche, les tâches sont subdivisées en sous-tâches assignées à chaque sous-système robotique selon les capacités de chacun (bras manipulateur ou base mobile).

- Le niveau communique avec un niveau inférieur dans lequel il y aura appel aux informations transmises des capteurs du robot.
- L'exécution proprement dite se fera au niveau le plus inférieur.

3.2 La planification hiérarchique :

La planification HTN (Hierarchical Task Network) est l'une des techniques qui est la plus utilisée en pratique pour des raisons d'efficacité mais aussi pour l'expressivité des langages HTN qui permettent de spécifier dans les domaines de planification des connaissances métier de haut niveau d'abstraction pouvant être utilisées par les algorithmes sous-jacents pour guider de manière efficace la recherche d'un plan solution. Contrairement à la planification classique ou le but est défini comme un ensemble de propositions à atteindre, en planification HTN le but s'exprime comme une tache ou un ensemble de taches à réaliser auxquelles il est possible d'associer des contraintes. Ce couple taches contraintes est appelé un réseau de taches. La recherche d'un plan solution consiste à décomposer le réseau de taches initiales définissant le but, en respectant les contraintes spécifiées, en un ensemble de sous-taches primitives qui peuvent être exécutées par une action au sens classique de la planification. La décomposition est réalisée en appliquant des règles de décomposition définies par des opérateurs hiérarchiques de planification appelés méthodes. Chaque méthode définit une Décomposition possible d'une t ache en un ensemble de sous-taches avec les contraintes qui les lient. La décomposition se termine lorsque le processus de décomposition aboutit à un réseau de taches contenant uniquement des taches primitives exécutables par une action et dont ' l'ensemble des contraintes associées sont vérifiées.

3.3 Algorithme de planification HTN :

La fonctionnent de base d'un planificateur HTN est décrit par les points suivants :

- En entrée : problème de planification p.
- Si P contient des tache primitives, alors résoudre les conflits et retourner le résultat.

- Choisir une tâche complexe t dans l'ensemble p.

- A partir de la base définit choisir une extension pour la tache t
- Remplacer t par son extension.

- Trouver des interactions entre les taches de p est suggérer des façons de les gérer.

3.4 Exigence d'un planificateur HTN :

Afin de concevoir un planificateur htn, il est nécessaire de prendre en Considération les trois points suivants [27]:

- La sémantique : il n'existe pas de sémantique bien définie pour la décomposition de tâche, cela a pour conséquence de rendre difficile tout jugement de la complétude ou de la consistance d'un plan.
- Conception : la conception de ce type de planificateur nécessite de prévoir et d'analyser toutes les tâches possiblement existantes et décomposables. Il est vraiment difficile d'une part, de produire la liste exhaustive de toutes .les décompositions d'une tâche, et d'autre part, à chaque ajout de fonctionnalité au système, il faudra prévoir et lister toutes les nouvelles décompositions à ajouter pour utiliser au mieux les fonctionnalités du système.
- Fragilité : les planificateurs HTN sont incapables de prendre en compte des tâches non explicitement prévues par le concepteur, même si les tâches élémentaires sont suffisantes pour construire un plan correct.

3.5 Formalise d'un planificateur HTN :

Définition 1 :

Un problème HTN est un quadruplet P = (s0, w, O, M) ou` s0 est l'état initial défini par un ensemble de propositions qui caractérisent le monde, ´ w est un réseau de taches initial qui définit le but, O est un ensemble d'opérateurs qui définissent les actions qui peuvent être r réalisées, et M un ensemble de méthodes qui définissent les décompositions possibles d'une tache composée en taches primitives [2].

Définition 2 :

Un opérateur est un triplé o = (name(o), pre(o), eff(o)) ou` name(o) Est une expression syntaxique de la forme t (u1, ..., uk) ou t est le nom de l'operateur et u1, ..., uk sont des variables ou des constantes typées qui définissent les paramètres de l'operateur. ´ pre(o) définit le pré conditions de l'operateur qui doivent être vérifiées pour le déclencher. eff(o) sont les effets de l'operateur qui définissent les propriétés générées par l'operateur. ´ pre(o) et eff(o) sont représentés sous la forme d'une expression ´logique. Leurs sous-ensembles pre+(o), pre−(o), eff+(o), eff− (op) représentent des sous-ensembles positifs ´ et négatifs. Une action à est un opérateur totalement instancie qui définit une fonction de transition permettant de ´ passer d'un état ´ s a un ` etat ´ S0 comme suit: s 0 = ((s\eff − (a)) ∪eff + (a)). A est applicable dans un état si pre+(a) ⊆ s et pre−(a)∩s = ∅. Toutes les formules atomiques d'une action sont totalement instanciées et appelées, par abus de langage, propositions [27].

Définition 3 :

Une méthode est un triplé m = (name(m), subtasks(m), constr(m)), ou` name(o) est, comme pour un opérateur, une expression syntaxique de la forme t (u1, ..., uk) ou t est le nom de la méthode et u1, ..., uk sont des variables ou des constantes typées qui sont utilisées dans la méthode. Une ou plusieurs méthodes peuvent de composer la même tache t(m). Chaque méthode représente une façon différente de la réaliser. subtasks(m) est l'ensemble de taches qui composent t(m) avec un tag qui les remplace dans le reste de la méthode. constr(m) est l'ensemble des contraintes qui portent sur subtasks(m). Le couple (subtasks(m), constr(m)) est représenté par un réseau de taches[27].

Définition 4 :

Une tache est une expression de la forme t (u1, ..., uk), ou t est le nom de la tâche et u1, ..., uk. L'ensemble de ses paramètres. Lors de la définition du domaine ces paramètres peuvent être soit des variables soit des constantes. Une méthode ou un opérateur sont dit pertinents pour une t ache t si t est égal ´ a` name(m) ou t est egal ´ a` name(o). Il existe deux types de taches: (1) Des taches composées qui peuvent être décomposées en sous ´ taches en appliquant une méthode de décomposition, (2) des taches primitives définies par des opérateurs et qui ne peuvent pas être décomposées [27]..

Définition 5 :

Un réseau de taches est le tuple tn = (U, C), ou U est un ensemble des taches et C un ensemble de contraintes qui portent sur ces taches. Un réseau de taches ne contenant que des taches primitives est dit primitif et représente un plan solution si toutes les contraintes qu'il contient sont satisfaites. Dans une tache on peut définir des contraintes [27] :

- une contrainte d'ordre est une expression de la forme (séries t1, t2, ..., tk). Elle signifie que dans un plan solution, la tache t1 doit être ordonnée avant la tache t2, t2 avant t3 ainsi de suite jusqu'à tk
- une contrainte du type Before est une expression de la forme (before (ϕ) (t1, ..., tk)) ou` ϕ est une expression logique, et (t1, ..., tk) est une liste de taches. Les contraintes Before servent à vérifier que l'expression ´ ϕ est vraie dans l'état juste avant la première tâche du groupe (t1, ..., tk).
- les contraintes du type After s'écrivent comme des contraintes Before sous la forme d'une expression (after (ϕ) (t1, ..., tk)) et signifient que ϕ doit être vraie dans l'état juste après` l'exécution de la dernière tache de (t1, ..., tk).
- une contrainte Between est une expression de la forme de (between (ϕ) (t11, ..., t1k) (t21, ..., t2k)). L'expression logique ϕ doit être vérifié dans tous les états entre la dernière tache de (t11, ..., t1k) et la première de (` t21, ..., t2k).

3.6 PDDL :

Conçu pour permettre une représentation commune des problèmes de planification pour la première compétition internationale de planification **[28]** le langage a connu depuis de nombreuses évolutions. Ce langage et les compétitions associées ont permis de fédérer les recherches en planification classique, Nous décrivons ci-dessous l'évolution du langage PDDL devenu un standard, lié aux compétitions successives :

PDDL 2.1

Version utilisée pour la compétition de 2002 [Long and Fox, 2003]. Elle introduit deux extensions majeures : la prise en compte du temps, et les variables numériques. La finesse de prise en compte des aspects temporels se situe à mi-chemin de la représentation complexe à base de chroniques temporelles utilisée par exemple dans les planificateur IxTeT [Ghallab and Laruelle, 1994] et ASPEN [Chien et al., 2000] ou plus récemment dans le cadre CNT (« Constraint Networks on Timelines ») [Verfaillie et al., 2010, Pralet and Verfaillie, 2010], et de la représentation simplifiée du planificateur ZENO [Smith and Weld, 1999] étendant la relation d'exclusion mutuelle de GRAPHPLAN [Blum and Furst, 1997]. En PDDL, les préconditions des actions dont on définit la durée d'exécution doivent être vérifiées à l'instant de début ou de fin d'une action, et/ou dans l'intervalle de temps ouvert compris entre ces instants ; les effets sont produits soit à l'instant de début, soit à l'instant de fin. La concurrence temporelle entre actions est possible à condition qu'à tout instant de l'exécution d'un plan, un même atome ne soit à la fois produit ou requis par une action et retiré par une autre.

La concurrence peut même être nécessaire pour l'obtention d'un plan dans certains problèmes, comme en témoignent divers travaux sur la notion de planification dite temporellement expressive [Cushing et al., 2007a,b] pour laquelle la complexité théorique du problème de l'existence d'un plan est EXPSPACE-complet [Rintanen, 2007]. Les variables numériques sont des quantités dont une valeur (minimale, maximale ou exacte) est requise en pré condition, et qui sont modifiées par l'exécution de l'action par augmentation, diminution ou

assignation. Ces variables numériques servent notamment à représenter des ressources, notion couramment utilisée dans le domaine de l'ordonnancement [Laborie and Ghallab, 1995, Laborie, 2003]. Le langage permet aussi de définir une fonction objectif sur la qualité du plan, exprimée comme une combinaison linéaire de la durée globale d'exécution du plan (le makespan) et des variables numériques. L'extension de PDDL 2.1 dénommée PDDL+ [Fox and Long, 2006] inutilisée lors des compétitions étend les notions temporelles et numériques de PDDL 2.1 à la modélisation de processus continus, permettant entre autres la représentation d'évènements exogènes et une prise en compte plus fine de la modification des variables numériques (vue comme un processus continu linéaire) augmentant la possibilité de concurrence entre les actions [28].

Description, générale du langage PDDL :

Le langage PDDL est un langage de codage standard pour les tâches de planification "classiques"

Composantes d'une tâche de planification PDDL [28] **:**

- Objets : Les choses dans le monde qui nous intéressent.
- Prédictions : Propriétés des objets qui nous intéressent ; peut-être vrai ou faux
- Etat initial : L'état du monde dans lequel nous commençons.
- Spécification de l'objectif : Ce que nous voulons être vrais.
- Spécification de l'objectif : Ce que nous voulons être vrais.

Les tâches de planification spécifiées dans PDDL sont séparées en deux fichiers :

- Fichier de domaine : Pour les prédicats et les actions.
- Fichier de problème : Pour les objets, l'état initial et la spécification des objectifs.

Fichiers de domaine

(define (domain<domain name>)

<PDDL code for predicates>

<PDDL code for first action>

[...]

<PDDL code for last action>

)

Fichier de problème

(define (problem<problem name>)

(: domain<domain name>)

<PDDL code for objects>

<PDDL code for initial state>

<PDDL code for goal specification>

)

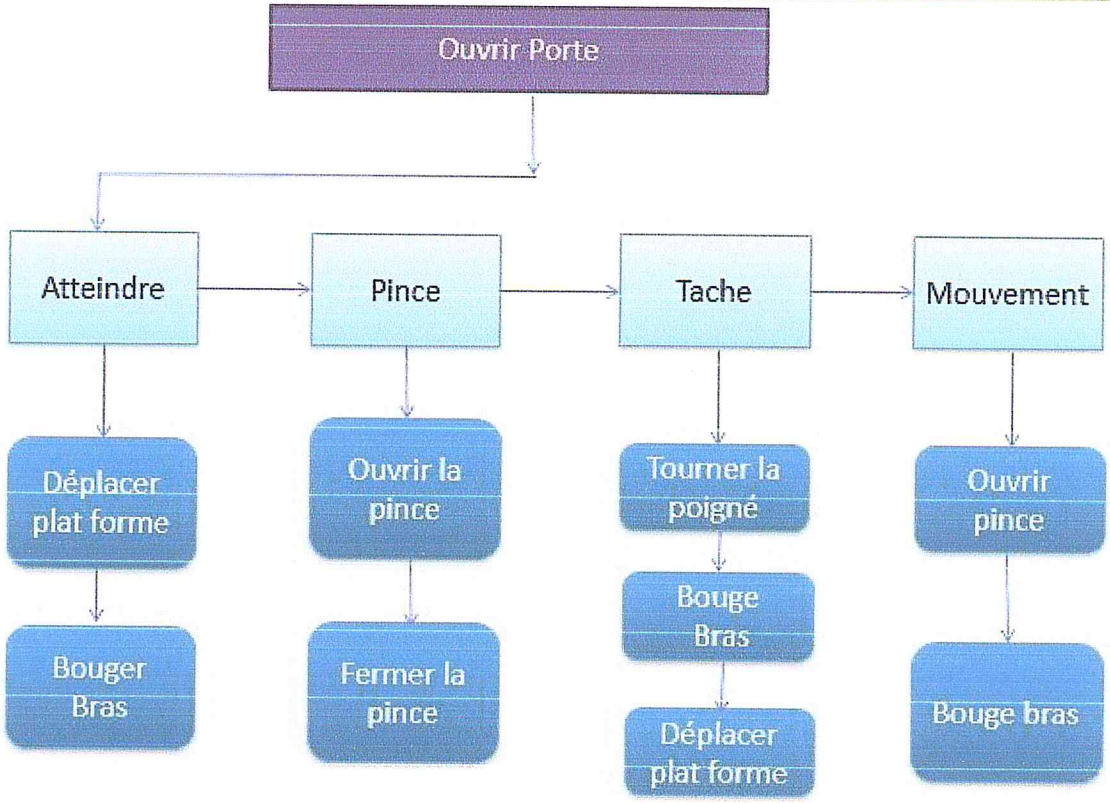
3.7 Présentation de notre solution en HTN

Dans le cadre de notre travail nous allons appliquer la planification sur les taches suivants :

3.8.1 Ouvrir porte :

Le diagramme ci-dessous décrit la décomposition de la tâche complexe ouvrir porte en des sous-taches primitives, cette décomposition est faite par un niveau hiérarchique, en passant du niveau plus haut (complexe) au bas niveau (primitive).

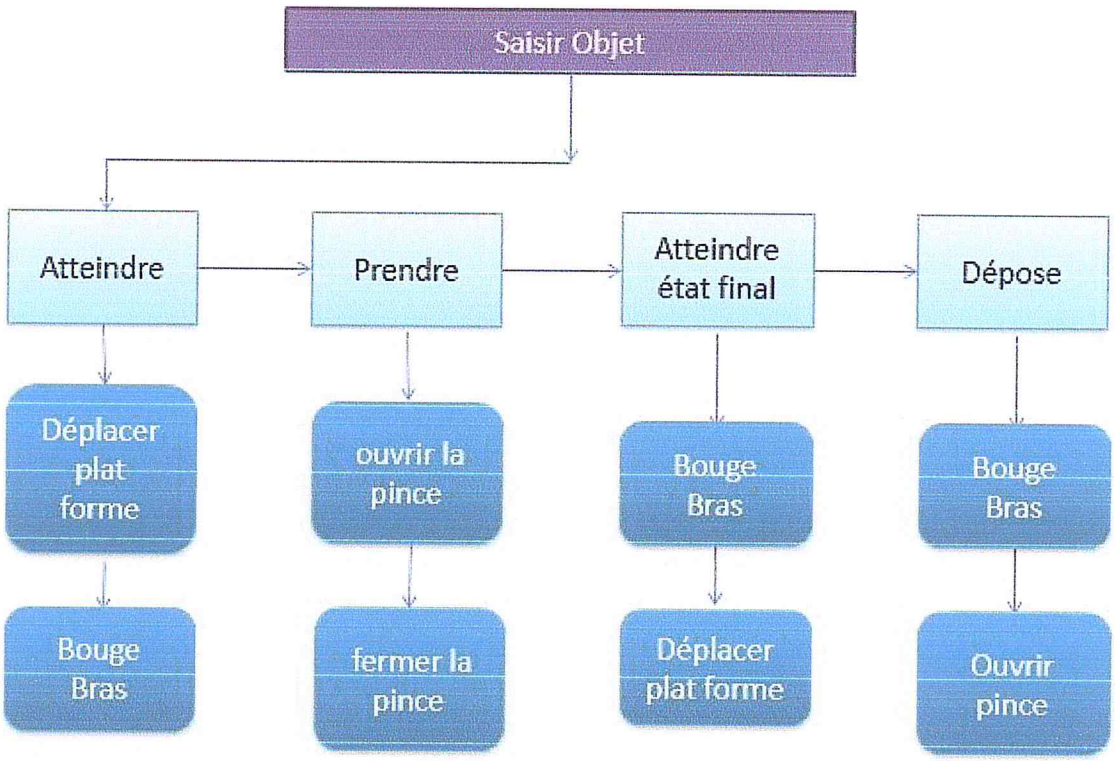
Le graphe de séquence de la tâche ouvrir porte.



3.8.2 Saisir objet :

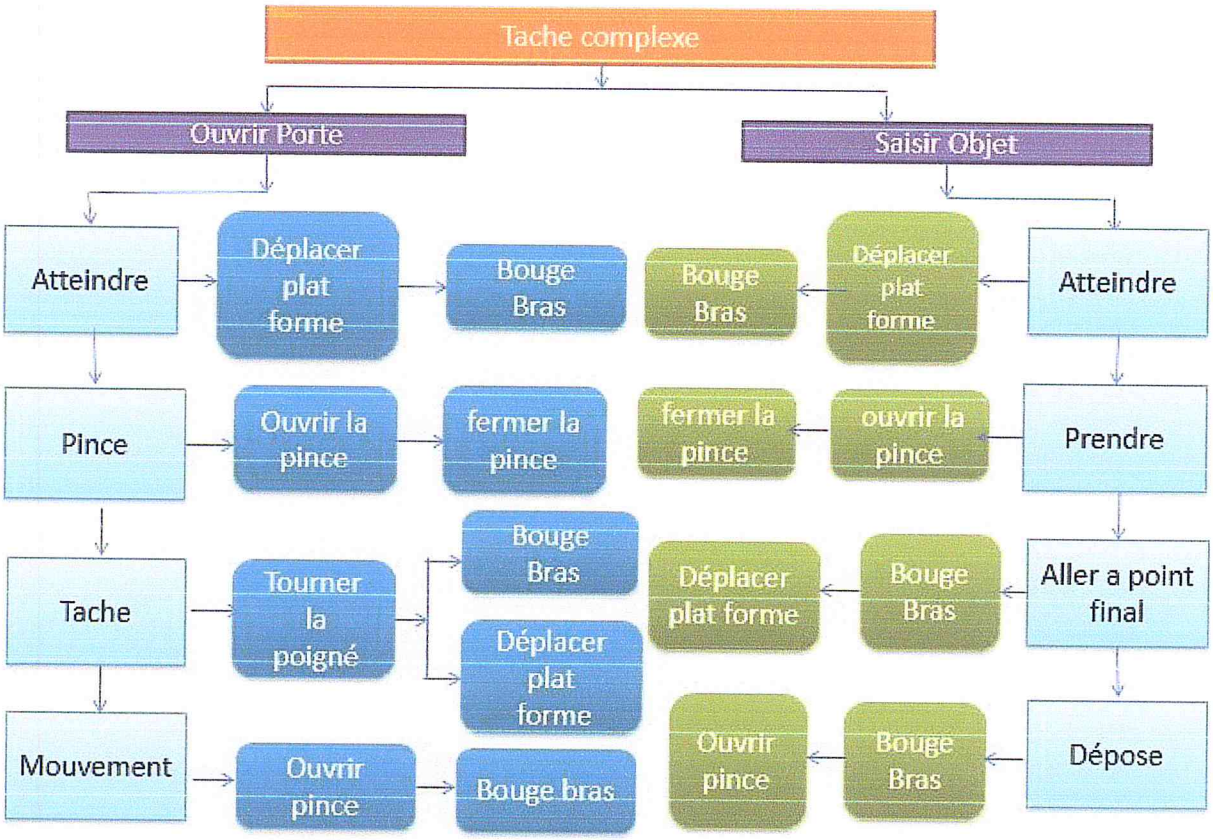
Le diagramme ci-dessous décrit la décomposition de la tâche complexe saisir objet en des sous-taches primitives, cette décomposition est faite par un niveau hiérarchique, en passant du niveau plus haut (complexe) au bas niveau (primitive).

Le graphe de séquence de la tâche saisir objet.



3.8.3 Tâche complexe :

Le diagramme ci-dessous décrit la décomposition de la tâche complexe qui combine les deux taches ouvrir porte et saisir objet en des sous-taches primitives, cette décomposition est faite par un niveau hiérarchique, en passant du niveau plus haut (complexe) au bas niveau (primitive).



3.8 Présentation de notre solution en PDDL :

3.9.1 Exemple saisir objet en PDDL

```
(define (problem saisair)
    (:domain saisair_objet)
    (:objects objet location1 location2 gripper)
    (:init (objet ob ) (location l1 ) (location l2 ) (grippr g)
    (at_robby l1) (at_local ob l1)  (free g) )
    (:goal (at_local ob l2))
```

```
            )
(define (domain <saisair_objet>)
   (:predicates ((objet ?ob) (location ?l)(Gripper ?g)
            (at-robby ?ob) (at_local ?ob ?l) (free ?g)  )
   (: action prendre
     :parametres (?ob ?l ?g)
     :precondition(and (objet ?ob) (location ?l) (gripper ?l)
                        (at_local ?ob ?l) (at_robby ?l) (free ?g))
    :effect (and (carry ?g ?ob )(not (at_local ?ob ?l))(not (free ?g)))
  )
  (:action depose
    :parametres (?ob ?l ?g)
    :precondition(and (objet ?ob) (location ?l) (gripper ?l)
                                    (carry ?g ?ob) (at_robby ?l))
    :effect(and (at_local ?ob ?l) (free ?g) (not (carry ?g ?ob)))
  )
  (:action deplacement
     :parametres (?r –robot ?from ?to -location)
     :precondition (and (adjacent ?from ?to)
                        (At_robby ?from)  (not (occupied ?to))
```

:effect (and (at_robby ?to) (not (occupied ?from))

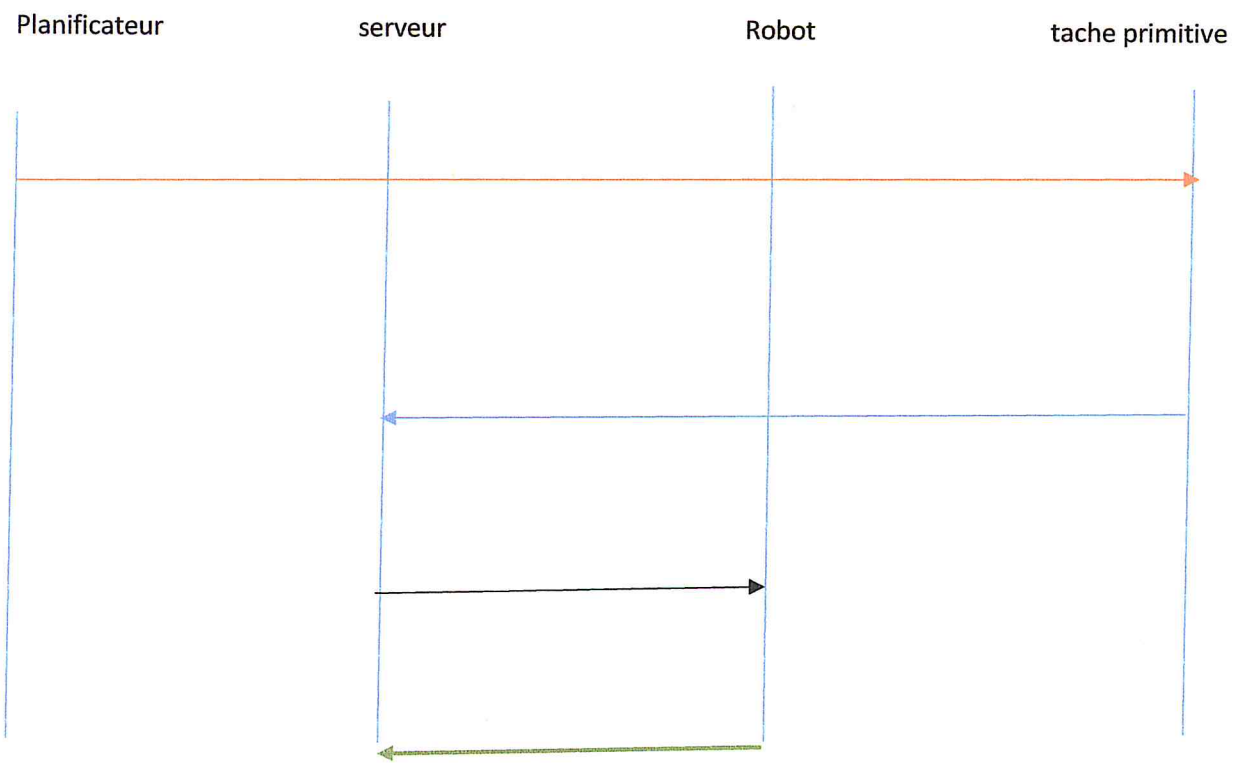
(occupied ?to) (not (at_robby ?from))

)

)

3.10 Diagramme de communication :

Le schéma ci-dessous décrit le fonctionnement de notre programme, en premier temps, le planificateur choisit une tache primitive qui doit être exécutée. En second temps, la tache primitive doit être chargée au serveur. Ensuite, le serveur communiquera avec le robot. A la fin, le robot répond par l'exécution de cette tache primitive.



3.11 Conclusion :

Au cours de ce chapitre, nous avons fait une description de notre solution qui est basée sur la planification hiérarchique d'un point de vue conceptuelle, de plus on a décrit les exemples en s'appuyant sur cette méthode de planification.

Dans le chapitre suivant, nous allons faire une étude sur les outils utilisée pour accomplir ce travail et les résultats qu'on a eu à la fin de notre réalisation.

Chapitre 4
Implementation

4.1 Introduction

Dans ce dernier chapitre qui consiste aux test et expérimentations, nous allons faire une description des outils matériels et logiciels qu'on a utilisés dans la réalisation de notre travail :

Matériels : Robot de notre organisme d'accueil CDTA.

Logiciels : ROSPLAN.

Ensuite nous allons donner les résultats qu'on a aboutis à la fin de notre projet.

4.2 Outils utilises

4.2.1 Materiels:

4.2.1.1 **Roboter/ulm :**

Le robot mobile RobuTER est une plate-forme automatisée et programmable de transport d'objets de taille moyenne. Grâce à son pilotage en différentiel de vitesse, le RobuTER peut tourner sur lui-même. Il est équipé d'un bras Ultraléger (ULM).voir figure 4.1

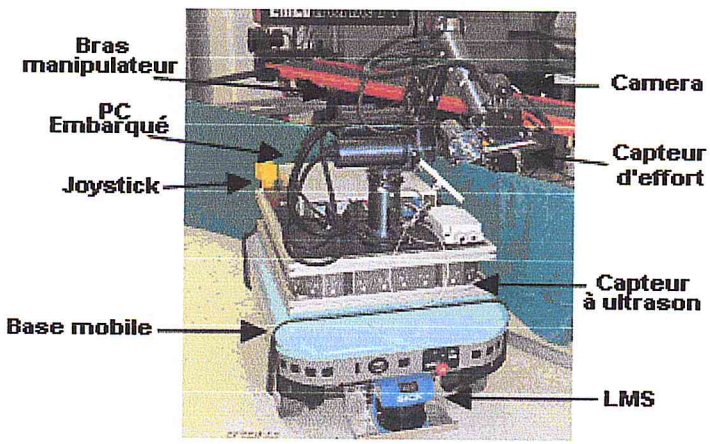


Figure 4.1 une représentation du robuter /uml.

4.2.1.2 **Le robot mobile Robuter :**

La plateforme mobile RobuTER est composée de quatre roues. Son poids est approximativement égal à 150kg, et une capacité de portée de 15 kg. Les roues sont actionnées par des moteurs électriques à courant continu. Elles permettent une grande mobilité sur sols lisses. Deux autres roues non actionnées (roues folles) assurent la stabilité de la plateforme mobile. Cette dernière est également alimentée par quatre batteries de 12 Volts chacune, lesquels apportent l'énergie suffisante nécessaire pour un mouvement autonome. La plateforme est équipée d'un capteur LMS et d'une ceinture de capteurs à ultrasons. Une carte à contrôleur (MPC555) assure les mouvements des moteurs de type « MBR CF8035F2 ».voir figure 4.2

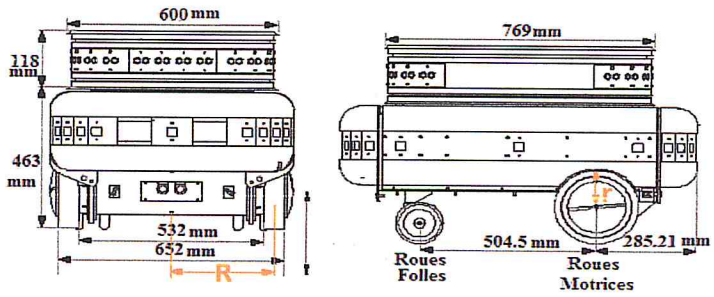


Figure 4.2 dimension de la plateforme mobile robuter

4.2.1.3 **Bras manipulater :**

Le bras manipulateur est composé de six articulations. Il est équipé d'une pince électrique à deux doigts assurant des tâches de saisie d'objets. Elle opère avec un contrôle tout ou rien (ouverture /fermeture) avec un signal de contrôle à deux états. La capacité de charge du système articulé est de 2kg lorsque le bras est totalement déployé. Le bras est composé de 7 moteurs à courant continu pour les six articulations rotoïdes et l'ouverture/fermeture de la pince. Le mouvement de chaque articulation est détecté par un capteur incrémental. Une butée articulaire électrique permet de sécuriser le mouvement de chaque articulation, afin de limiter les mouvements angulaires. La portée du bras est de 700 mm avec une répétitivité de +/- 1 mm. Voir figure 4.3

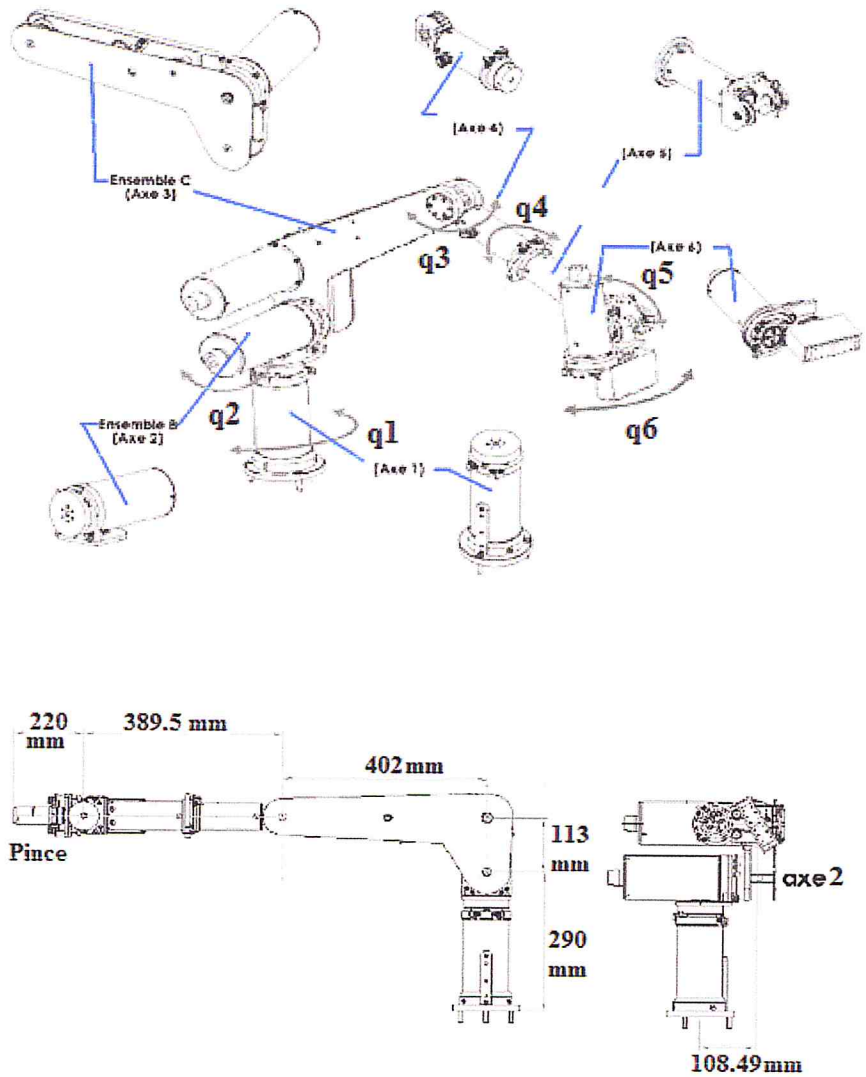


Figure 4.3 Bras manipulateur ULM(a)Dimensions des axes (b) Disposition des articulations

Le bras manipulateur est composé de trois premières articulations composées des moteurs «PARVEX RS320H ». Les autres articulations se meuvent grâce aux moteurs de type « Harmonicdrive RH-11D-3001 ». Ils délivrent une précision de mouvement et une grande capacité de couple.

I	q_i^{min}	q_i^{max}
1	-94°	94
2	-22	87
3	0	90
4	-77	77
5	-73	38
6	-50	144

Figure 4.4 Les paramètres d'un robot manipulateur mobile.

4.2.1.4 **Principaux capteurs du Roboter /ulm :**

Le robot manipulateur mobile est doté d'un certain nombre de capteur citant entre autre :

- Capteurs ultrason
- Capteur lms sick 2000
- Capteurs incrémentaux
- Capteur de fin de course

4.2.1 **Logiciels:**

4.2.2.1 **Définition :**

ROSPlan est un framework logiciel intégré avec ROS pour permettre l'application d'algorithmes de planification de tâches basés sur le langage PDDL 2.1 dans ROS. Le frame work ROSPlan (Cashmore et al. 2015) fournit une méthode générique pour la planification des tâches dans un système ROS. ROSPlan relie deux standards - PDDL2.1 et ROS - encapsulant à la fois la planification et l'expédition. Il possède une interface simple, et certains composants pour communiquer avec les bibliothèques ROS standard [29]. Voir figure 4.5

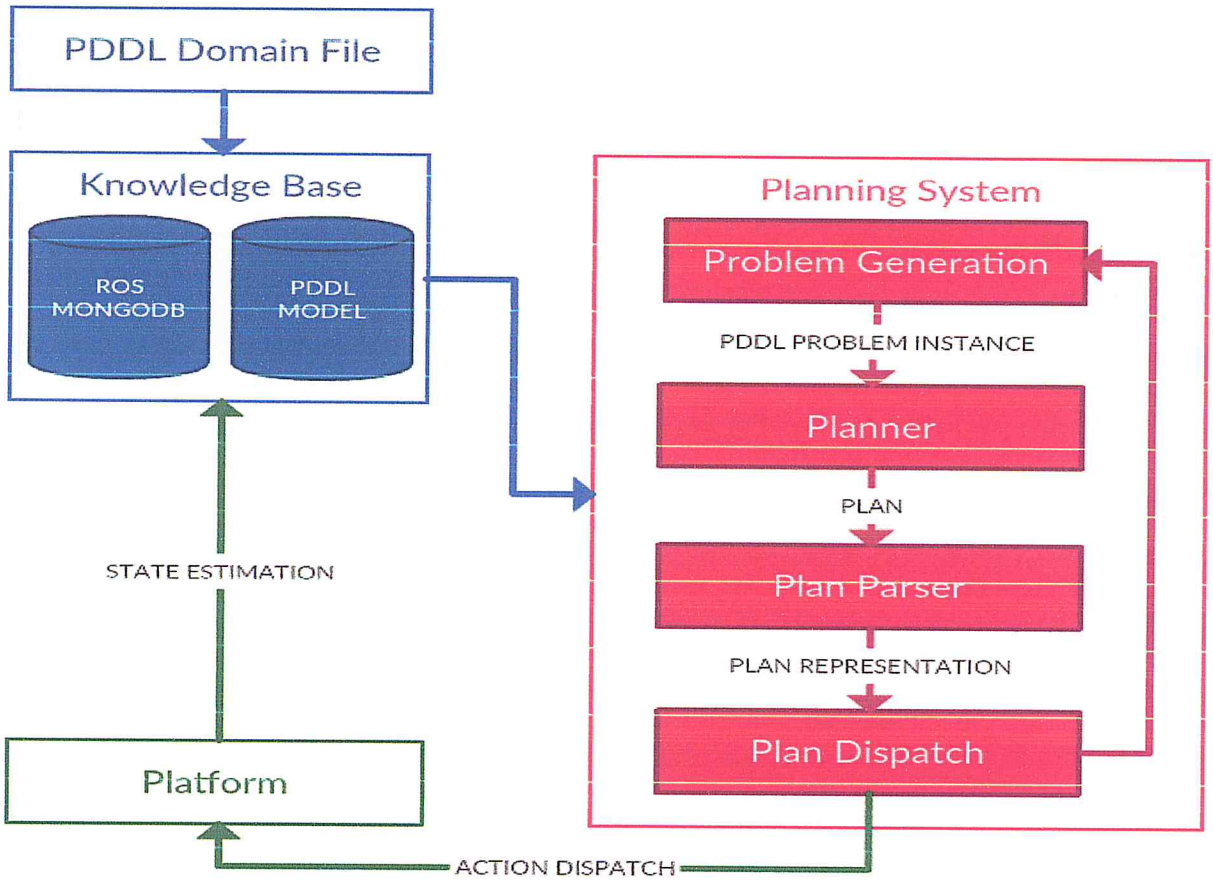


Figure 4.5 Architecture ROSPlan

4.2.2.2 Base de connaissance

- Elle est utilisée pour stocker modèle pddl
- Elle stocke à la fois un modèle de domaine et l'instance d'un problème en cours

Principe de fonctionnement : on peut résumer le fonctionnement de cette base de connaisse en ces trois points suivants :

- Charge un modèle pddl a partir du fichier.
- Stocke l'état comme une instance pddl.
- Mise a jour par des messages ROS

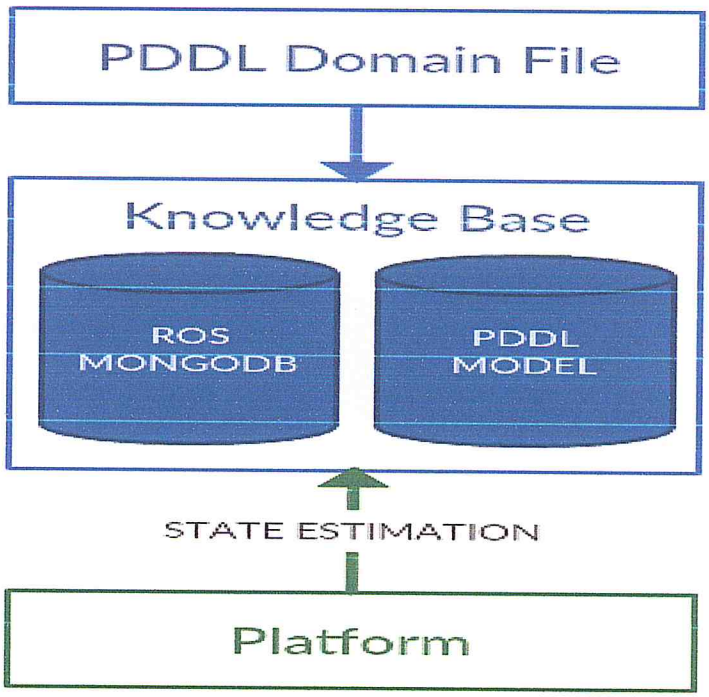


Figure 4.6 Base de connaissance

La plupart du temps, il y aura également des informations à stocker qui ne font pas partie du modèle PDDL. Pour cela nous utilisons ROS mongoDB par défaut.

4.2.2.3 Systéme de planification

Le système de planification s'intègre avec le planificateur, au moins qu'il ne soit pas modifiée, ces principales fonction sans dans ce décrites dans les points suivantes [29] :

- Extraire les informations de la base de connaissances et générer un fichier de problème pddl.
- Appeler un planificateur et traiter un plan de sortie
- Transmet le plan via des messages ros.

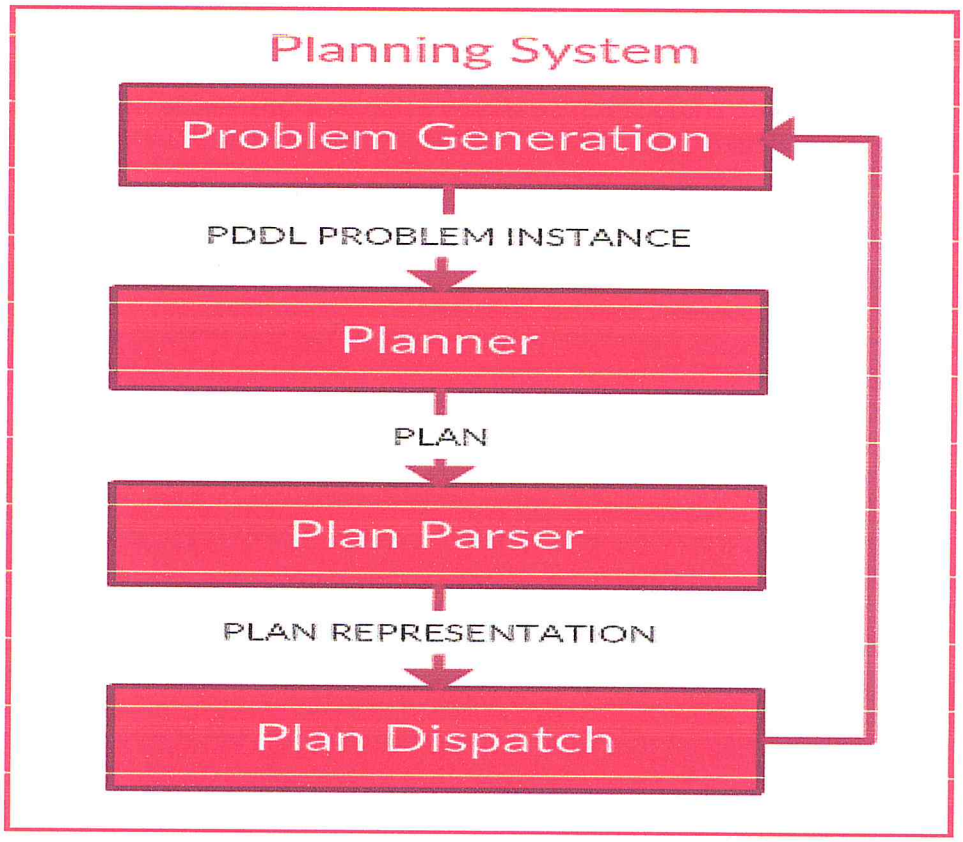


Figure 4.7 système de planification

4.2.2.4 Lancement du système de planification :

Il existe deux façons pour lancer le système de planification, cette diversité est due à la fonctionnalité requise [29] :

4.2.2.4.1 Lancement de l'interface planner :

- Lancement de l'interface de planification fournit des services pour générer des instances de problème et pour appeler le planificateur [29].
- Le plan n'est pas expédié [29].
- L'interface de planification lance un nœud unique qui fournira des services pour la génération et la planification de problèmes. Il peut être lancé en ajoutant le nœud suivant à votre fichier de lancement [29] :

```
<!-- domain file -->
      <param name="domain_path" value="$(find
rosplan_demos)/common/domain_turtlebot.pddl" />

<!-- planning system -->
<node name="rosplan_planner_interface" pkg="rosplan_planner_interface"
type="plannerInterface" respawn="false" output="screen">
    <!-- directory for generated files -->
        <param name="data_path" value="$(find rosplan_config)/planner/" />
    <param name="problem_path" value="$(find
      rosplan_config)/planner/problem.pddl" />
    <param name="strl_file_path" value="$(find
      rosplan_config)/planner/plan.strl" />

    <!-- to run the planner -->
    <param name="planner_command" value="timeout 10 $(find
      rosplan_config)/planner/bin/popf -n DOMAIN PROBLEM" />
</node>
```

Figure 4.8: interface planner est décrite dans un fichier XMl, dans le l'extension est .launch

4.2.2.4.2 Lancement du système de planification :

Le lancement du système de planification, inclut l'interface de planificateur, mais aussi inclut l'expédition automatique du plan. Pour lancer le système de planification, il faut ajouter le nœud décrit par le fichier XML suivant à votre fichier de lancement :

```
<!-- domain file -->
    <param name="domain_path" value="$(find
     rosplan_demos)/common/domain_turtlebot.pddl" />
<!-- planning system -->
      <node name="rosplan_planning_system" pkg="rosplan_planning_system"
       type="planner" respawn="false" output="screen">
<!-- directory for generated files -->
      <param name="data_path" value="$(find rosplan_config)/planner/" />
<param name="problem_path" value="$(find rosplan_config)/planner/problem.pddl"/>
      <param name="strl_file_path" value="$(find rosplan_config)/planner/plan.strl" />
      <!-- to run the planner -->
      <param name="planner_command" value="timeout 10 $(find
        rosplan_config)/planner/bin/popf -n DOMAIN PROBLEM" />
      <param name="generate_default_problem" value="true" />
        <param name="max_dispatch_attempts" value="2" />
</node>
```

Figure 4.9. Fichier xml décrivant le nœud qui permet de lancer le système de planification

4.2.2.3 Paramètres :

- Domain-path : chemin d'accès au fichier pddl [29].
- Data-path : chemin d'accès au fichier généré automatiquement tels que le fichier plan [29].
- Problem -path : chemin d'accès a l'instance du problème généré automatiquement [29].
- Planner-command :ligne de commande utilisée pour exécuter planner.les chaines « DOMAINE »et « PROBLEM » seront remplacées par des chemins vers modèle de

domaine pddl el les fichiers d'instance de problème respectivement. La ligne par défaut ci-dessus exécute le planificateur POPF, avec le flag"-n", et un délai d'attente de 10 secondes [29].

- Max_dispatch_attemps : le nombre de fois ou le système réapparaitra et retournera l'envoi en cas d'échec du plan [29].
- Generate_default_problem : Si vrai, une nouvelle instance de problème pddl2.1, sera générée à chaque tentative de planification. Elle est générée à partir du modèle pddl stocké dans la base de connaissance. Si faux, un problème ne sera pas généré, cela est utile si la ligne de commande n'exécute pas un planificateur ou si l'instance de problème est générée par d'autres moyens [29].

4.2.2.4 Utilisation de l'interface planner :

L'interface planificateur effectue les fonctions suivantes :

- Généré l'instance du problème PDDL, par l'extraire du modèle pddl à partir de base de connaissance [29].
- Appellee le planificateur [29].
- Lit et stocke la partie du planificateur [29].

4.2.2.5 Composant :

4.2.2.5.1 Service de génération de problème :

Le service de génération de problème vous permet d'utiliser l'interface du planificateur pour générer un problème à partir de la base de connaissances, sans planification. Il est appelé de la façon suivante [29] :

Topic: */kcl_rosplan/problem_server_params*
Service: *rosplan_dispatch_msgs/PlanningService*

Un problème sera généré à partir de la base de connaissances. Le problème sera publié sur topic du problème. Le service retourne lorsque le problème est généré. Certains paramètres de lancement sont annulés par la requête de service: domain_path, path_proble, data_path.

4.2.2.5.2 Services de planification :

Le processus de planification générera automatiquement un fichier de problème, appellera le planificateur à l'aide du paramètre de ligne de commande et analysera la sortie du planificateur. Le problème et le plan sont publiés comme décrit dans la liste des topic.il existent 4 étapes pour lancer le système de planification [29] :

- Appeler le service du système de planification de base
- Apple au service du système de planification avec paramètres
- Envoi d'un but d'action au serveur d'action du système de planification
- Utilisation d'une interface graphique rosplan

a) **Service de système de planification de base :** Le système de planification commencera. Le service retourne lorsque [29] :

- Le planificateur ne parvient pas à trouver un plan
- La dépêche se termine avec succès
- Il n'y a plus de tentatives de ré-planification
- Ou le système de planification rencontre une erreur

Topic:*/kcl_rosplan/planning_server_params*

Service: *rosplan_dispatch_msgs/PlanningService*

b) Planification du service système avec paramètres :

Comme le service de base. Certains paramètres de lancement sont remplacés par

La requête de service: *domain_path, problem_path, data_path, planner_command.*

Topic:*/kcl_rosplan/start_planning/goal*

Message: *rosplan_dispatch_msgs/PlanActionGoal* [29].

c) Envoi d'un objectif d'action au système de planification :

Topic:*/kcl_rosplan/start_planning/goal*

Message: *rosplan_dispatch_msgs/PlanActionGoal*

Démarrer le système de planification en tant qu'action ROS [29].

4.3 Travailler avec ROS

Ces points suivant résument les principales fonctions de ros plan :

4.3.1 Remplacement de planificateur :

Le planificateur par défaut dans ROSPlan est une version de POPF, prenant en charge la planification à tout moment pour PDDL2.1. Ce planificateur est pour les domaines avec des contraintes temporelles et aucune incertitude. Il y'a trois étapes pour remplacer le planificateur[29] :

a) Fournir une ligne de commande pour appeler le nouveau planificateur :

Système de planification ROSPlan appelle planificateur en utilisant une ligne de commande qui peut être modifiée dans fichier de lancement :

```
<!-- planning system -->

<node name="rosplan_planning_system" pkg="rosplan_planning_system"
type="planner" respawn="false" output="screen">
<!-- other parameters removed -->
<!-- planner command line -->
<param name="planner_command" value="timeout 10 $(find
rosplan_planning_system)/common/bin/popf -n DOMAIN PROBLEM" />
</node>
```

"DOMAIN" doit être remplacé par le chemin d'accès du fichier de domaine, et "PROBLEM" doit être remplacé par le chemin du fichier d'instance de problème.

b) Analyser la sortie du nouveau planificateur :

La sortie d'un planificateur doit être analysée dans une structure de données pour l'expédition. Par exemple, les actions du plan peuvent être stockées dans une liste pour une exécution séquentielle. La structure de données que vous utilisez dépendra de la méthode d'envoi du plan. Inclus dans ROSPlan est l'expédition séquentielle ; Et temporelle, concurrente, contingente comme Esterel. La sortie du planificateur est enregistrée comme un fichier et

également publiée sous forme de message ROS. Le fichier est enregistré dans un emplacement spécifié par un paramètre dans le fichier de lancement :

```
<!-- planning system -->
<node name="rosplan_planning_system" pkg="rosplan_planning_system"
type="planner" respawn="false" output="screen">
<!-- other parameters removed -->
<!-- directory for generated files -->
<param name="data_path" value="$(find rosplan_planning_system)/common/" />
</node>
```

Un fichier "plan.pddl" sera généré dans data_path.

Le plan est analysé par la classe : **rosplan_planning_system::PlanParser**

c) Générer une instance problématique :

L'instance du problème est (re) générée avant chaque tentative de planification. Il est généré en demandant un modèle à jour à partir de la base de connaissances. Vous pouvez désactiver la génération d'un fichier par défaut par défaut en utilisant un paramètre de lancement :

```
<!-- planning system -->
<node name="rosplan_planning_system" pkg="rosplan_planning_system" type="planner" respawn="false"
output="screen">
        <!-- other parameters removed -->
  <!-- to run the planner -->
  <param name="generate_default_problem" value="false" />

</node>
```

Le problème par défaut généré est un problème de planification PDDL2.1. Le problème est généré par : ***Rosplan_planning_system ::PDDLProblemGenerator*** [29].

4.3.2 Remplacement de la génération de problème :

Le problème par défaut généré est un problème de planification dans PDDL 2.1 Le problème est généré par : rosplan_planning_system :: PDDLProblemGenerator.

La génération de problème par défaut utilise un fichier de domaine et les informations stockées dans le système de gestion des connaissances. Si vous souhaitez modifier la génération de problème par défaut, il y a une étape non plus :

- Désactiver complètement la génération de problème par défaut;
- Ou Fournir une nouvelle implémentation de PDDLProblemGenerator

4.3.3 Replacement de l'ajout du plan :

L'envoi par plan par défaut est une implémentation de rosplan_planning_system : PlanDispatcher. Il existe deux implémentations par défaut [29] :

rosplan_planning_system :: SimplePlanDispatcher

rosplan_planning_system :: EsterelPlanDispatcher.

Une nouvelle implémentation de cette classe peut être donnée pour fournir différents comportements d'envoi, ou pour envelopper un contrôleur de répartition existant.

La manière dont un plan est expédié peut être complètement échangée. Si vous souhaitez remplacer l'envoi du plan, il existe une ou deux étapes:

- Fournir une nouvelle implémentation de rosplan_planning_system :: PlanDispatcher
- En option, fournir une nouvelle représentation du plan [29].

La nouvelle implémentation de rosplan_planning_system : PlanDispatcher peut être une interface avec un système d'envoi et d'exécution existant [29].

4.3.4 Ajout d'une action :

Par défaut, une action PDDL dans ROSPlan est envoyée comme un message ActionDispatch [29].

Topic Name: ***kcl_*** **rosplan*****/action_dispatch***

Message Type: rosplan***_dispatch_msgs/ActionDispatch***

De même, feedback sont publiés sous la forme d'un message ActionFeedback.

Topic Name: ***kcl_rosplan/action_feedback***

Message Type: ***rosplan_dispatch_msgs/ActionFeedback***

Pour lier une nouvelle action, il existe une des deux façons. Non plus:

- Écrivez un nouveau nœud qui utilise ces deux Topic
- Ou Écrivez une sous-classe de rosplan_action_interface :: ActionInterface

4.3.5 Ajout d'une estimation d'état :

L'ajout d'une estimation d'état est effectué en mettant à jour la base de connaissances. Ceci est décrit en détail dans l'utilisation de la base de connaissances et l'ajout à la base de connaissances. Pour ajouter une estimation d'état au système : écrivez un nœud ROS qui utilise les services [29].

4.4 Architecture de communication :

Comme on as vu dans le chapitre 1, un programme sous ROS fonctionne en se basant sur le principe des nœuds .dans le cadre de notre mémoire, on a utilisé les 3 nœuds suivants : voir figure 4.9

- Nœud pour planificateur.
- Nœud pour tache primitive.
- Nœud serveur qui permet de faire la communication au niveau du robot.

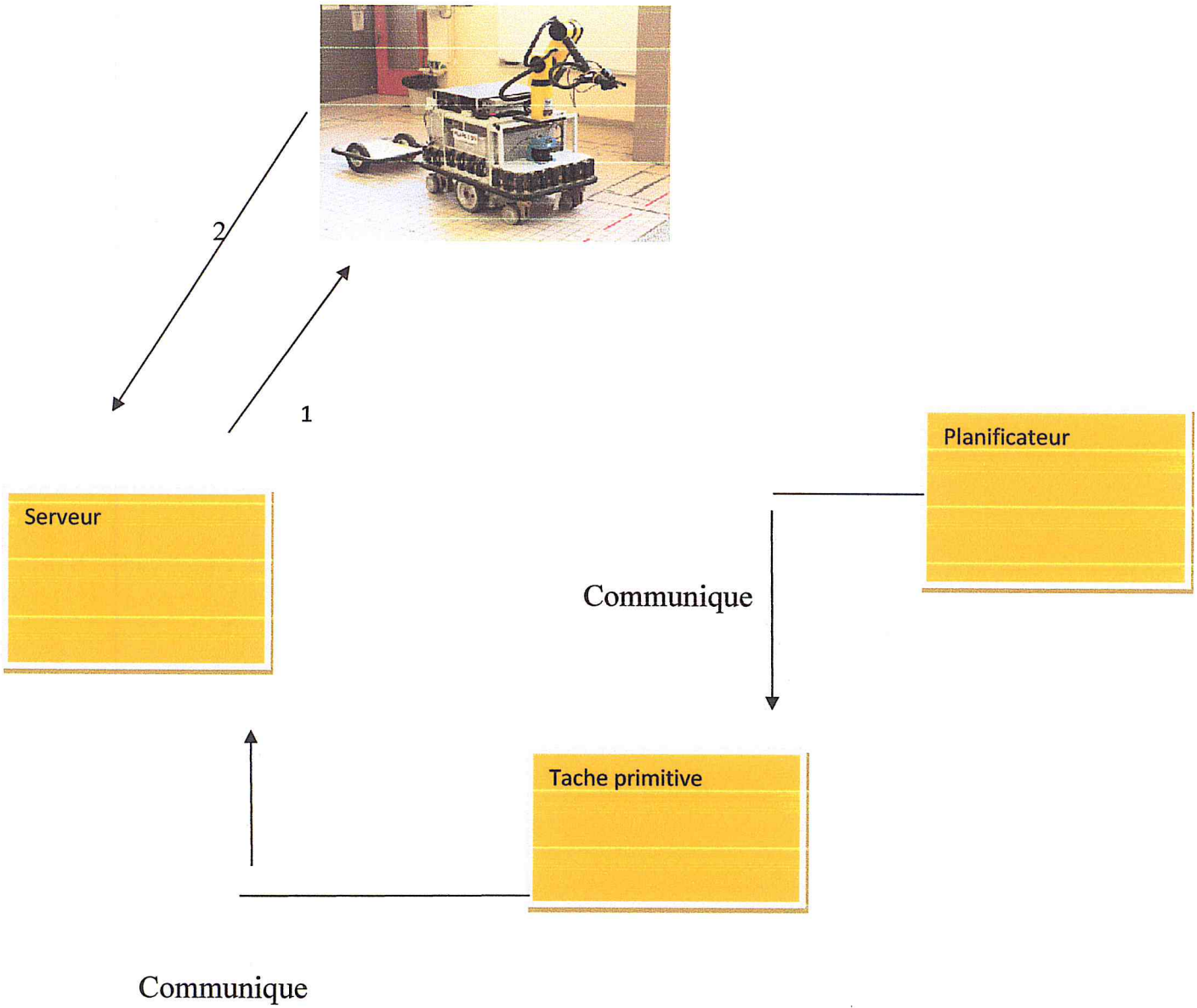


Figure 4.10Architecture de communication.

4.5 Résultat

- Lors de la simulation sur ROSPLAN, en premier temps on a eu des résultats voir figure 4.11, mais lorsqu'on relance la simulation une deuxième fois, on a eu un problème au cause d'une version non stable de ce frame wok. Pour cela, on a changé notre méthode de travail.

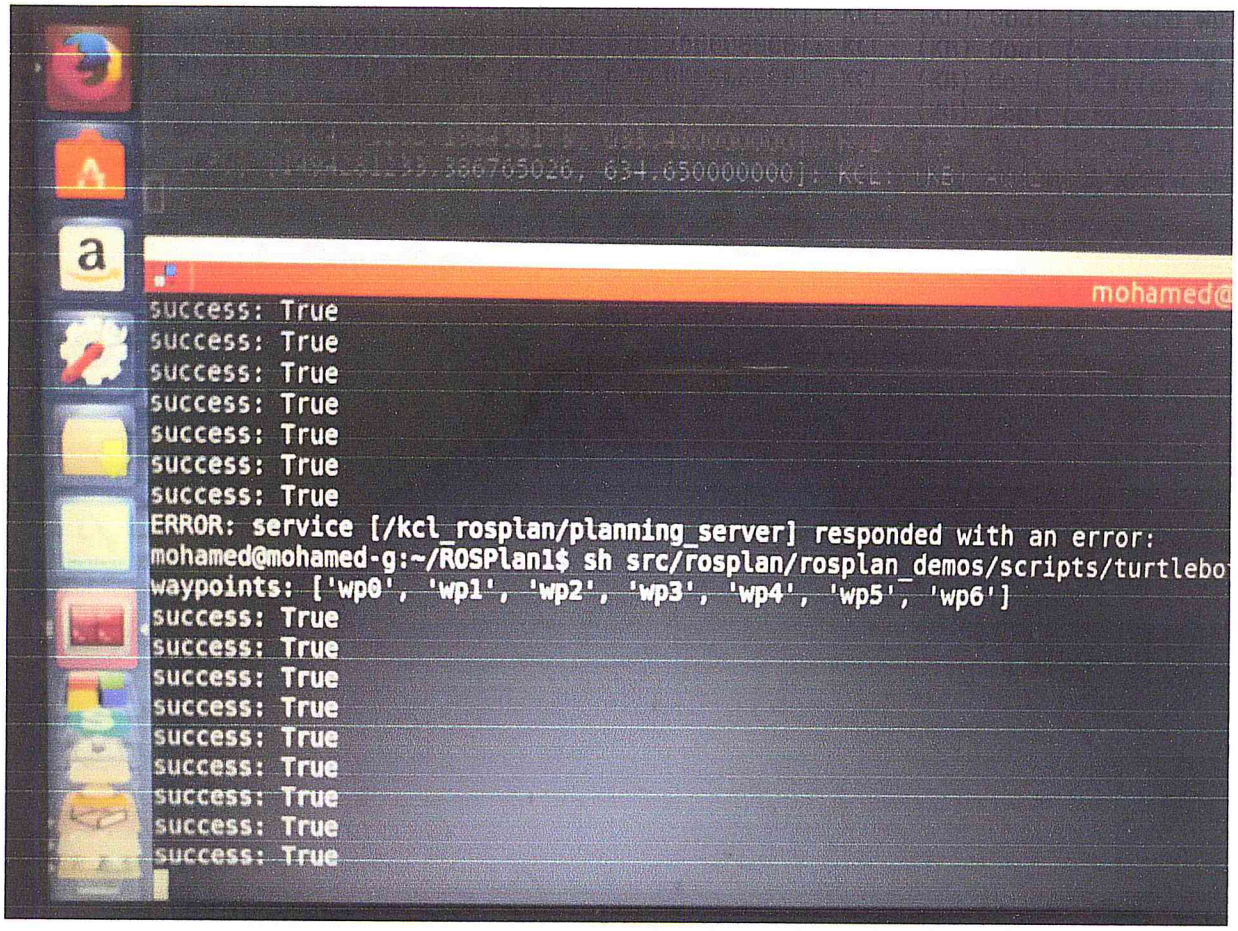


Figure 4.11 l'exécution d'un plan sous ROSPLAN.

- Après l'exécution de notre programme sur le robot réel, voici les résultats qu'on a eus :
  - Déplacer plate-forme :

Figure 4.12plate-forme avant déplacement

Figure 4.13 : plate-forme après déplacement.

➢ Bouger bras :

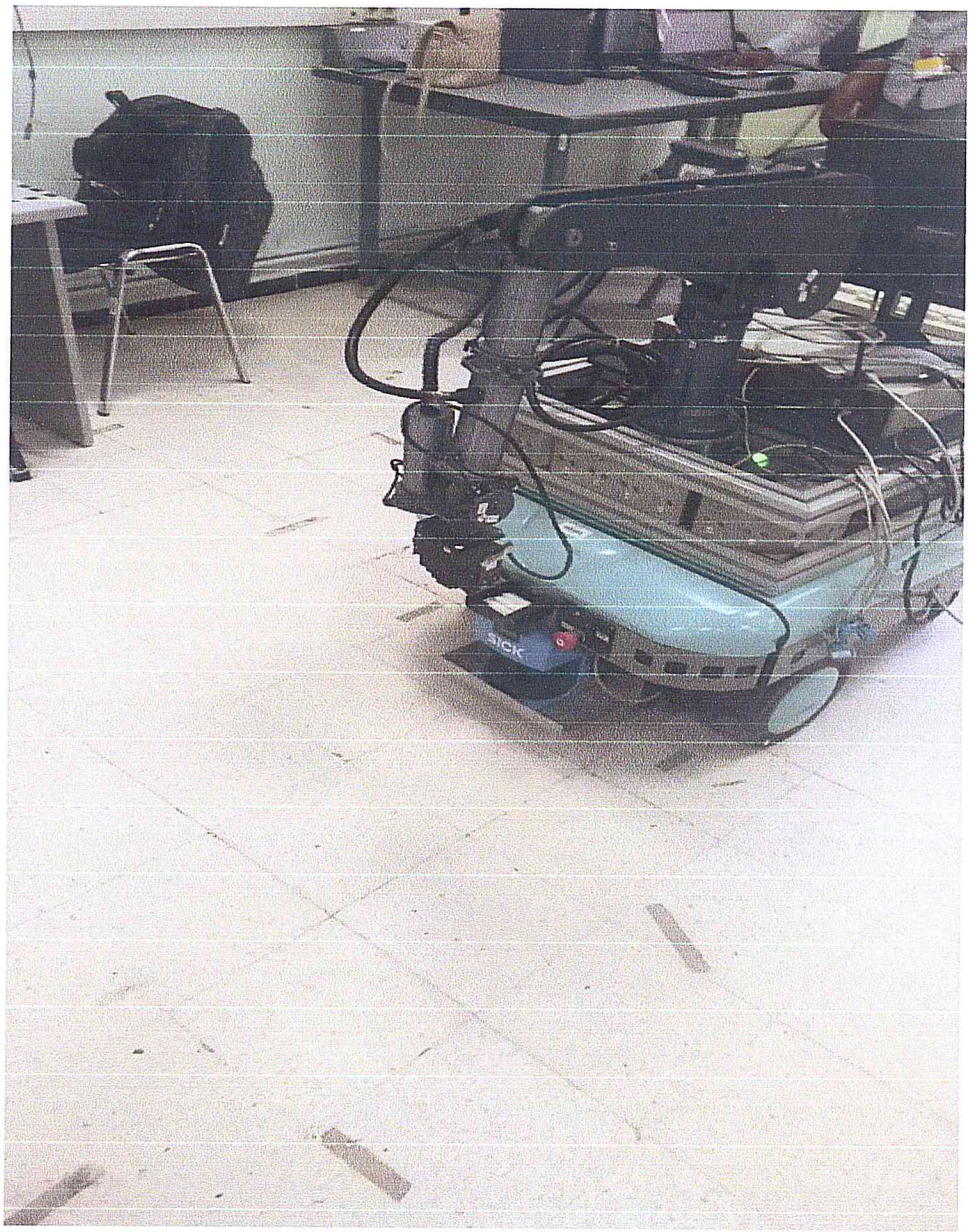


Figure 4.14 bras avant le mouvement bras.

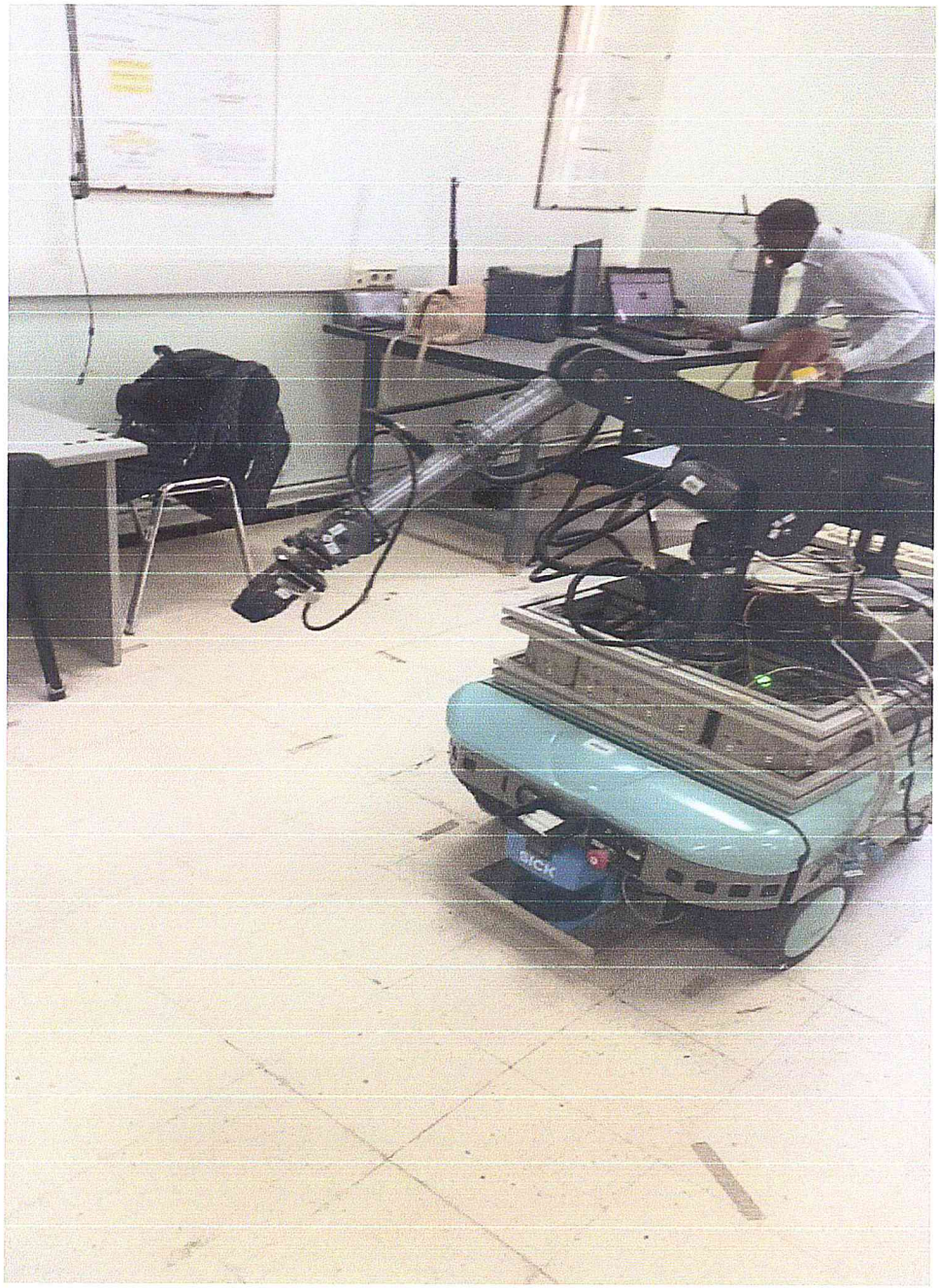


Figure 4.15 bras après le mouvement

4.6 Conclusion : Dans ce chapitre on a abordés, les outils qu'on a utilisés dans la réalisation de notre projet, ainsi les résultats qu'on a aboutis à la fin. Dans ce qui suit nous allons conclure notre travail par une conclusion générale.

Conclusion générale

Conclusion générale :

La réalisation d'un robot intelligent autonome et capable de réaliser des tâches complexes est un défi important à prendre. Parmi les nombreuses capacités devant être mise à contribution pour relever un tel défi, un robot doit avant tout être capable de prendre ses propres décisions et de planifier lui-même comment il vas réaliser ces missions.

Suite à une analyse de différente techniques de planification, nous nous sommes orientés vers la planification hiérarchique. Cette technique de planification permet de décomposer une tâche complexe en sous tâches primitives afin d'arriver à une tâche demandée ou accomplir le but donné

Dans notre mémoire, on a implémenté un programme de planification des tâches complexes et qui sont connus dans la littérature de planification des tâches pour un robot manipulateur mobile, ces deux taches sont nommées : ouvrir une porte et saisir un objet.

Dans le cadre de réalisation de notre projet on a utilisé des outils logiciels et des outils matériels : les premier tel que ROS, C++,ROSPLAN et le PDDL qui est un langage de spécification de problèmes. Les outils matériels utilisés sont le robuter ULM de notre organisme de stage CDTA et un serveur de communication. Avec l'élaboration de différentes techniques de planification et de décomposition de tâches, on est arrivé à réaliser un système de planification qui permet d'arriver à notre but souhaité qui correspond à ouvrir une porte ou saisir un objet.

A la fin de notre travail, on obtient un plan qui permet de décrire notre solution, en outre, on aimerait appliquer une fonction heuristique qui permet de choisir la meilleure solution pour arriver à l'état final vu que notre solution de planification peutt être décrite par une automate qui a un état initial et un état final.

Référence :

- [1] http://www.futura-sciences.com/tech/definitions/robotique-robotique-603/
- [2] http://www.futura-sciences.com/tech/definitions/robot-604/
- [3] http://www.larousse.fr/dictionnaires/francais/robot/69647
- [4] http://www.linternaute.com/dictionnaire/fr/definition/robot/
- [5] http://www.linternaute.com/dictionnaire/fr/definition/robot/
- [6] 1983 *Pensée et Machine*. Seyssel, Champ Vallon BENSAUDE-VINCENT, Bernadette
- [7] 2004 *Designing sociable robots*. Cambridge, MIT Press.BROOKS, Rodney
- [8] R.C. Arkin: A Behavior-based Robotics. MIT Press, Cambridge, MA, USA, 1998.
- [9] F. Bacchuset F. Kabanza: Using temporal logics to express search control knowledge for planning.Artificial Intelligence, 116(1-2):123–191, 2000.
- [10] http://wiki.ros.org/
- [11] E. Beaudry, Y. Brosseau, C. Côté, C. Raïevsky, D. Létourneau, F. Ka-banzaet F.Michaud: Reactive planning in a motivated behavioral architecture.DansNational Conference on Artificial Intelligence (AAAI), pages 1242–1249, 2005.
- [12] E. Beaudry, F. Kabanzaet F.Michaud: Planning for a mobile robot to attend a conference. Dans B.Kéglet G. Lapalme, éditeurs : Canadian Conference on AI, vol. 3501 de Lecture Notes in Computer Science, pages 48–52. Springer, 2005
- [13] M. Beetz, W. Burgard, D. Foxet A.Cremers: Integrating active localization into high-level robot control systems. Robotics and Autonomous Systems, 23:205–220, 1998.
- [14] A. Blumet M. Furst: Fast planning through planning graph analysis. Dans International Joint Conference on Artificial Intelligence (IJCAI), pages 1636–1642,1995.
- [15] R.A. Brooks: A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation, 2(1):14–23, 1986.
- [15] C. Côté, D. Létourneau, F. Michaudet Y.Brosseau: Software design pat-terns for robotics : Solving integration problems with MARIE. Dans Workshop of Robotic Software Environment, IEEE International Conference on Robotics and Automation, 2005.

- [16] E.W. Dijkstra: A note on two problems in connexion with graphs. Dans Numerische Mathematik, vol. 1, pages 269–271, 1959.
- [17] M.B. Doet S.Kambhampati: SAPA : A scalable multi-objective metric temporal planner.Journal of Artificial Intelligence Research (JAIR), 20:155–194, 2003.
- [18] A. Gereviniet I. Serina: LPG : A planner based on local search for planning graphs with action costs. Dans International Conference on Artificial Intelligence Planning Systems, pages 13–22, 2002.
- [19] Erolet al., 1994] Kutluhan Erol, James A. Hendler, and Dana S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. InProc. of the 2nd Int. Conf. on Artificial Intelligence Planning Systems (AIPS), pages 249–254. AAAI Press, 1994.
- [20] [Erolet al., 1995]Kutluhan Erol, Dana S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidabil-ity results for domain-independent planning. Artificial In-telligence, 76(1):75–88, 1995.
- [21] [Erolet al., 1996] Kutluhan Erol, James A. Hendler, and
- Dana S. Nau. Complexity results for HTN planning.An-nals of Mathematics and Artificial Intelligence, 18(1):69–93, 1996.
- [22] [Geier and Bercher, 2011] Thomas Geier and Pascal Bercher. On the decidability of HTN planning with task insertion. In Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI), pages 1955–1961. AAAI Press, 2011.
- [23] [H¨ olleret al., 2014] Daniel H¨ oller, Gregor Behnke, Pascal Bercher, and Susanne Biundo. Language classification of hierarchical planning problems. InProc. of the 21st Eu-rop. Conf. on Artificial Intelligence (ECAI), pages 447–452. IOS Press, 2014.
- [24] [Kambhampatiet al., 1998] Subbarao Kambhampati, Amol Mali, and Biplav Srivastava. Hybrid planning for par-tially hierarchical domains. In Proc. of the 15th Nat. Conf.on Artificial Intelligence (AAAI), pages 882–888. AAAI Press, 1998.
- [25] [Knuth, 1977] Donald E. Knuth. The complexity of songs. SIGACT News, 9(2):17–24, July 1977.
- [26] [Linet al., 2008] Naiwen Lin, Ugur Kuter, and Evren Sirin. Web service composition with user preferences. InProc. Of the 5th Europ. Semantic Web Conference (ESWC), pages 629–643, Berlin, Heidelberg, 2008. Springer.

- [27] [Nauet al., 2003]Dana S. Nau, Tsz-Chiu Au, Okhtay Il- ghami, Ugur Kuter, J William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system.Journal of Artificial Intelligence Research, 20:379–404, 2003.
- [28] M. Foxet D.Long: PDDL 2.1 : An extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research (JAIR), 20:61–124,2003.
- [29] https://github.com/KCL-Planning/ROSPlan/wiki

Annexes

ROS-ROSPLAN

Robot Operating System and ROSPLAN

1- Installation ROS Indigo
2- Création et Configuration WorkSpace , Package
3- Creation nodes Publisher et Subscriber (c++)
4- Execution la travail
5- Installation ROSPLan

1-Installation ROS Indigo

a - Spéciations des sources.list

sudo sh -c 'echo "deb http ://packages.ros.org/ros/ubuntuprecise main" > /etc/apt/sources.list.d/ros-latest.list'

b - Spéciations des clefs

wget http ://packages.ros.org/ros.key -O - j sudo apt-key add -

c - Mise à jour des listes de paquets

sudo apt-get update

d - Installation des paquets

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

2- Création et Configuration WorkSpace , Package

A - Ligne a ajouté au fichier .bashrc

source /opt/ros/indigo/setup.bash

B- Création de l'espace catkin

mkdir -p ~/catkin ws/src

cd ~/catkin ws/src

catkin init workspace

d- Création package

catkin_create_pkg My_package std_msgs rospy roscpp

Nous pouvons maintenant construire ce package en utilisant catkin_make :

cd ~/catkin ws/

catkin_make

source devel/setup.bacsh

3 - Creation nodes Publisher et Subscriber (c++)

Dans /catkin ws/src on écrit deux fichier Talker.cpp et listner.cpp et en suite ajouter ce code dans les fichiers.

Talker.cpp

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>
int main(int argc, char **argv) {
ros::init(argc, argv, "talker");
ros::NodeHandle n;
ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
ros::Rate loop_rate(1.0);
int count = 0;
while (ros::ok()) {
std_msgs::String msg;
std::stringstream ss;
ss << "salam alykom " << count;
msg.data = ss.str();
ROS_INFO("%s", msg.data.c_str());
chatter_pub.publish(msg);
ros::spinOnce();
loop_rate.sleep();
++count;
}
return 0;
}
```

listner.cpp

```
#include "ros/ros.h"
#include "std_msgs/String.h"
void chatterCallback(const std_msgs::String::ConstPtr msg) {
   ROS_INFO("I lisen : [%s]", msg->data.c_str());
}
int main(int argc, char **argv) {
ros::init(argc, argv, "listener");
ros::NodeHandle n;
ros::Subscriber sub = n.subscribe<std_msgs::String>("chatter", 1000, chatterCallback);
ros::spin();
return 0;
}
```

4- Execution le travail

Dans le fichier my_package/C MakeLists.txt Ajouter l'exécutable de fichier talker et listner :

```
add_executable(talker src/talker.cpp)
target_link_libraries(talker ${catkin_LIBRARIES})
add_dependencies(talker My_pakage_generate_messages_cpp)

add_executable(listener src/listener.cpp)
target_link_libraries(listener ${catkin_LIBRARIES})
add_dependencies(listener My_package_generate_messages_cpp)
```

Et après on lance les nœuds comme suite :

Catkin_make

roscore

rosrun My_package talker

rosrun My_package listener

5- Installation ROSPLan

- On ajoute la commande suivante dans terminal :

(pour Indigo) sudo apt-get install flex ros-indigo-mongodb-store ros-indigo-tf2-bullet freeglut3-dev

- Creation de WorkSpace

```
mkdir -p ROSPlan/src
cd ROSPlan/src
```

Apres dans notre workspace on installe les get suivantes :

git clone https://github.com/clearpathrobotics/occupancy_grid_utils
git clone https://github.com/KCL-Planning/rosplan
git clone https://github.com/KCL-Planning/rosplan_interface_turtlebot2

apres on ajoute la commande pour catkin notre workspace

```
source /opt/ros/indigo/setup.bash
catkin_make
```