Université Paris Est Créteil

UFR des sciences et de la technologie

Mémoire de Master 2

Systèmes complexes Technologies de l'Information et du Contrôle (ScTIC)

Thème

Coordination de robots pour le transport d'objets

Entreprise d'accueil :

Laboratoire Images, Signaux et Systèmes Intelligents LISSI

Sujet proposé par :

Mr A. HAZAN

Présenté par :

Imene BOUYOUCEF

Soutenu le 26/09/2013

Dédicace

Je dédie ce travail à

L'âme de mon père

Ma mère

Mes frères et mes sœurs

Mes amis et tous ceux que j'aime …

IMENE

Remerciements

C'est avec plaisir que je réserve ces quelques lignes en signe de gratitude et de profonde reconnaissance à tous ceux qui ont contribué à l'aboutissement de ce modeste travail.

Je remercie Monsieur Kurosh MADANI de m'avoir accueilli au sein du laboratoire LISSI et Monsieur Aurélien HAZAN pour sa disponibilité et son encadrement de qualité.

Je tiens aussi à remercier tous les professeurs de La formation et tous les membres de l'équipe au laboratoire LISSI.

Que les membres du jury trouvent ici l'expression de mon remerciement pour l'honneur qu'ils me font en acceptant de jurer mon modeste travail.

Table des matières

Liste des figures

Introduction générale :

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

Actuellement les systèmes multi-robots sont utilisés pour aider ou remplacer l'homme dans la réalisation de certaines tâches dans plusieurs domaines; tels que : la médecine, le domaine militaire, l'industrie etc…… . Parmi les tâches à réaliser dans le dernier domaine que nous avons cité est le transport des objets dans les usines, cette tâche fait l'objet de mon travail.

La thématique générale abordée dans ce projet est la coordination des robots pour le transport d'objets, cette thématique fait l'objet de recherches actives et de nombreux soucis se posent encore, la robustesse, la capacité d'apprentissage, la coordination, la communication.

Plus précisément l'objet de mon travail vise à coordonner les entités d'un système multi-robots et à établir une communication entre elles afin d'élaborer un scénario de type d'interaction entre les robots.

Le premier chapitre du rapport est dédié à la présentation de l'organisme d'accueil et présenter le contexte de mon travail, dans le second chapitre j'ai présenté les problématiques et les approches proposés pour la coordination d'un système multi-robots. Le dernier chapitre est concentré pour les techniques génies logiciels utilisés pour la simulation des scénarios.

Chapitre 1 : Contexte du projet

1.1 Introduction :

Le présent chapitre introduit notre projet intitulé « coordination des robots pour le transport d'objets », la première partie est dédiée à la présentation de l'organisme d'accueil, la deuxième partie est consacrée à la présentation de projet et le travail demandé ainsi que ses objectif.

1.2 Présentation de LISSI (Laboratoire Image, Signaux et Systèmes Intelligents :

1.2.1 Généralités :

LISSI a été créé en 2005 par la Faculté des Sciences et l'Institut Universitaire de Technologies de l'Université Paris-Est Créteil Val-de-Marne (UPEC) c'est le résultat de la fusion de trois laboratoires de l'Université Paris 12, le LERISS, le LIIA et l'I2S.
Il est impliqué dans de nombreuses activités de recherche et d'applications concernant l'intelligence artificielle, le traitement du signal, la robotique.

Figure 1-1 : Logo de LISSI

1.2.2 Les domaines de recherche :

Les travaux menés par l'équipe de LISSI touchent les domaines de la science, la technologie de l'information, des communications et des systèmes.
Ce laboratoire est structuré en de deux équipes :
- Traitement de l'Image et du Signal (TIS) : ciblée vers les domaines de l'ingénierie biomédicale. Elle regroupe les compétences sur l'analyse des signaux non stationnaires et les méthodes de segmentation, de mise en correspondance et de recalage en imagerie.
- Systèmes complexes, Traitement de l'Information et de la Connaissance (ScTIC) : ciblée vers l'étude et la mise en œuvre des systèmes complexes ambiants et communicants, elle

regroupe les compétences dans les domaines de la modélisation et du contrôle/commande des systèmes complexes par les approches liées à l'intelligence artificielle.
J'ai effectué mon stage au sein du l'équipe ScTIC dont le thème est « La robotique coopérative »

1.3 Présentation du projet :

1.3.1 Cadre générale de travail :

La thématique des systèmes multi-robots fait l'objet de recherches actives mais de nombreux problèmes se posent encore, notamment concernant la robustesse, la capacité d'apprentissage et de réaction à des situations imprévues.
Le laboratoire LISSI et son équipe Machines et Systèmes Intelligents travaillent depuis de nombreuses années sur ces sujets, notamment sur la thématique de la perception. L'objectif de stage est d'´etudier la coordination dans les systèmes multi-robots pour le transport et la manipulation d'objets simples.

1.3.2 Travail demandé :

Dans le cadre de ce stage il est demandé de réaliser une étude bibliographique sur le transport d'objet et la manipulation dans un cadre multi-robots, pour mettre à jour la base bibliographique disponible au LISSI, élaborer un scénario type d'interaction. Par exemple : un robot fixé en un point B pose un objet sur un robot mobile. Le robot mobile déplace l'objet vers un point C où l'objet est récupéré. Par la suite programmer ce scénario en simulation sur le logiciel ROS. Proposer une mesure quantitative de robustesse de la tâche, porter le code sur une plate-forme robotique réelle. Le tester, et comparer ses performances avec celles obtenues en simulation, puis celles décrites dans la littérature, notamment les travaux précédents réalisés au LISSI et finalement publier le code et le rapport technique associé en libre accès

1.4 Système multi robot :

Le domaine de la robotique distribuée a ses origines dans les années 1980, lorsque plusieurs chercheurs ont commencé à étudier les questions dans plusieurs systèmes de robots mobiles. Avant cette date, la recherche s'est concentrée sur les systèmes robotiques simples qui n'impliquaient pas les composants robotiques de résolution de problèmes [3].

Un système multi robot est un système constitué d'un ensemble de robots, homogènes ou hétérogènes, mobile ou fixe, ces robots sont capables de communiquer entre eux et coopérer pour améliorer l'efficacité d'exécution des tâches ou encore pour permettre d'exécuter des tâches impossibles à exécuter de façon individuelle.

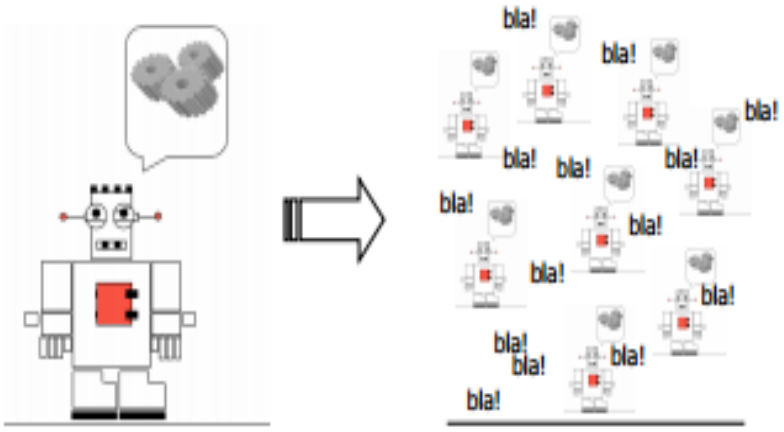

Figure 0-2 : Système Multi-Robots

1.5 Pourquoi un système multi robots :

Un système multi robots peut effectuer des tâches difficiles ou même impossibles à accomplir par un seul robot. Une équipe de robots fournit une certaine redondance, elle contribue à l'accomplissement d'une tâche de manière collaborative

Ce système offre plusieurs avantages par rapport aux systèmes mono-robot ; les motivations les plus communes pour le développement de solutions de systèmes multi robots sont les suivants : [2]

- La complexité de la tâche à réaliser est trop élevée pour un seul robot donc elle nécessite une coopération d'un ensemble de robots.

- La construction de plusieurs robots simples avec des ressources bornées est beaucoup plus facile que la construction d'un seul robot complexe et puissant.

- L'introduction de plusieurs robots augmente la robustesse grâce à la redondance.

- La rapidité : pour obtenir un niveau de performance élevé les tâches sont effectuées en parallèle. A titre d'exemple : l'exploration parallèle d'un environnement

inconnu par un ensemble de robots mobiles, dans le but de faire soit une cartographie de l'environnement, soit la réalisation d'une tâche de fourragement. Ces deux tâches peuvent être réalisées par un seul robot, mais l'ajout d'autres robots va faire en sorte d'accélérer l'exécution des tâches en question.

- La robustesse : les performances du contrôle dans un SMR ne sont pas trop affectées en cas de défaillance d'un robot.

- La flexibilité : il est possible d'exécuter les tâches désirées, de diverses manières. Ceci est induit principalement par la redondance des entités robotiques.

1.6 Les domaines d'utilisation de système multi robots :

L'un des objectifs de l'utilisation des robots est de remplacer ou d'aider les êtres humains dans la réalisation des certaines tâches. C'est particulièrement souhaitable pour des tâches dangereuses comme l'exploitation spatiale et sous-marine, l'exploration de l'intérieur d'une centrale nucléaire ou encore ennuyeuses tel que la surveillance d'un site, les applications militaires. Les robots peuvent effectuer des tâches automatiquement, mais ils sont aussi dotés d'une certaine intelligence.

1.6.1 Robotique médicale :

La robotique se développe et se perfectionne au grand bénéfice de la médecine, un robot médical est un appareil utilisé soit pour former les médecins, pour aider les médecins dans leurs tâches ou les remplacer ; en chirurgie les robots peuvent remplacer les chirurgiens qui les contrôlent à distance et aussi pour remplacer un organe du corps d'un malade, en général c'est un appareil aide à soigner des patients.

1.6.2 Robotique industrielle :

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

1.6.3 Robotique militaire :

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

1.7 Thèmes de recherche dans les systèmes multi robots :

Les chercheurs ont défini sept thèmes de recherche primaires dans les systèmes multi-robots [2] :

- les systèmes d'inspirations biologiques : la plupart des SMR font suite aux travaux des chercheurs qui se sont intéressée à la modélisation des sociétés d'insectes ou d'animaux et ont reproduit avec succès leurs comportements.

- Les systèmes qui étudient la communication : ces systèmes traitent le problème de communication soit entre robots ou bien entre un utilisateur et robots.

- Les systèmes qui s'intéressent aux architectures, l'allocation de tâches et le contrôle : les problèmes de ces systèmes sont communs aux systèmes décentralisés : allocation et planification de tâches, structure du système de communication, homogénéité ou hétérogénéité des robots.

- Les systèmes orientés vers la localisation, la cartographie, l'exploration.

- Les systèmes dans le but est le transport et la manipulation d'objet : ces systèmes proposent une méthode de planification du mouvement d'une équipe de robots pour le transport collectif d'un objet.

- Les systèmes de coordination de mouvements : ces systèmes étudient la planification des trajectoires des robots, la génération et le maintien de la formation ainsi que le contrôle de trafic ;

- Les systèmes dont l'objet est la conception de robots reconfigurables.

Dans notre travail nous nous intéressons aux systèmes multi-robots dont le but est la coordination pour le transport d'objet

1.8 Conclusion :

A travers ce chapitre nous avons mis le projet dans ce contexte de travail et présenté le cadre d'accueil, dans le chapitre suivant nous allons aborder les problématiques de coordination dans un système multi-robots.

Chapitre 2 : la coordination dans les systèmes multi-robots

2.1 Introduction :

Comme nous avons mentionné dans le chapitre précédent nous nous intéressons aux systèmes multi robots dont le but est le transport et la manipulation d'objet. Dans ce chapitre nous allons voir comment coordonner un ensemble de robots pour transporter un objet.

2.2 Transport d'objets par un système multi-robots :

L'objectif dans ce type de tâches est de transporter un objet par un ensemble de robots qui peuvent être homogènes ou hétérogènes.

2.2.1 Système multi-robots homogènes :

Un système multi-robots homogènes est un ensemble de robots qui ont les mêmes caractéristiques, ils doivent coopérer entre eux ; ils s'auto-organisent pour soulever, porter pousser et déposer l'objet qu'un seul robot ne pourrait réaliser tout seul.

Des travaux dans ce domaine ont été réalisés, on peut citer par exemple les travaux de Stiwell : plusieurs robots porteurs qui se positionnent en dessous d'une palette chargée pour la déplacer. Les robots mobiles disposent d'un capteur de force qui leur sert d'information pour contrôler le déplacement de la palette d'une manière stable et distribuée [2]

2.2.2 Systèmes multi-robots hétérogènes :

Un SMR hétérogènes est un ensemble de robots mixtes, c'est-à-dire les robots ne sont pas identiques, il peut y avoir des robots fixes et des robots mobiles ; dans ce cas chaque robot s'occupe de sa propre sous tâche, les robots manipulateurs fixes s'occupent du dépôt et le soulèvement d'objet et les robots mobiles s'occupent de son transport.

La problématique traitée dans le présent rapport est comment faire pour coordonner un ensemble de robots de ce dernier type?

2.3 Les approches méthodologiques pour faire coopérer et coordonner un ensemble de robots :

Des travaux précédents ont proposé trois approches traitant l'étude d'un ensemble d'entités autonomes, la première approche est orientée « sciences du vivant : éthologie des animaux sociaux », la deuxième approche émanant de l'informatique est intitulée « système multi agents », et la troisième approche issue de l'automatique est appelé « robotique collective » [2].

2.3.1 Approche des sciences vivantes :

Cette approche est inspirée de la fascinante organisation, des règles de contrôle et du comportement des diverses sociétés biologiques, tels que les fourmis, les oiseaux ou bien les abeilles, pour contrôler un groupe d'agents coopératifs, par exemple : la construction de nids, et la récolte de pollen chez les abeilles, le transport coopératif d'objets et le fourragement chez les fourmis etc... Cette dernière tâche consiste à rechercher la nourriture dans l'environnement et à la transporter par la suite au nid, elle était une source d'inspiration de travaux en robotique, et cela en remplaçant les fourmis par les robots, la nourriture par des objets dans l'environnement, les phéromones par des communications entre les robots et le nid par une position dans l'environnement. Des travaux en ce sens ont démontré la capacité des équipes multi-robot à affluer, se disperser, fourrager et suivre l'itinéraire.

Cette approche est exploitable dans le cas d'un SMR homogènes.

Figure 2-1 : tâche coopérative de transport d'objets par un groupe de fourmis

Figure 2-2 : tâche coopérative de transport d'objets par un groupe de robots mobiles

2.3.2 Approche informatique : système multi agents :

Des travaux en Intelligence Artificielle ont donnés naissance à l'intelligence Artificielle Distribuée (IAD) qui traitent l'intelligence produite par un ensemble d'agents coopératifs ; de l'IAD a émergé par la suite la notion de Système Multi Agent (SMA) qui résout des systèmes complexes en les décomposant en un ensemble d'agents autonomes.

Un exemple qui caractérise à la fois le monde de l'informatique et celui de la coopération d'agents, est celui du génie logiciel. Ce domaine produit des logiciels complexes en faisant coopérer et coordonner un ensemble d'agents logiciels : classe, objets, ou une technologie basée sur une conception orientée objets. Cette démarche tend à décomposer le système complexe en plusieurs modules (objets), c'est de même pour le système multi robots c'est-à-dire décomposer la tâche principale en sous tâches.

Cette approche peut être utilisée dans le cas d'un SMR homogènes ou hétérogènes

2.3.3 Approche automatique (robotique collective) :

Faire coopérer un ensemble de systèmes robotiques est un synonyme d'existence de modèles régissant le système multi robots. C'est ce qui est appliqué dans le cadre de la robotique coopérative, où chaque robot est modélisé de telle sorte à ce qu'il intervienne dans le modèle global du système.

2.4 Les problèmes de la coordination dans les systèmes multi-robots :

Un système multi-robots est considéré comme un système distribué, puisque les tâches sont distribuées sur l'ensemble des robots du système. L'un des problèmes des systèmes multi-robots est la coordination, ce problème est séparé en quatre parties, la communication, la computation, la configuration, et la coordination [4].

La communication : définir comment les agents communiquent entre eux.

La computation : elle définit la mise en œuvre du comportement des agents individuels. elle détermine ainsi ce qui est communiqué.

La configuration : elle définit la structure d'interaction et elle indique quel sont les agents existants dans le système et quel sont les agents qui peuvent communiquer entre eux.

La coordination : elle définit les modes d'interaction, c'est-à-dire déterminé à quel moment certaines communications ont lieu.

Dans mon travail j'ai focalisé mon intention sur le problème de la communication entre les entités du système multi-robots

2.4.1 Comment communiquer les robots :

Communiquer est un synonyme de partage de l'information, d'expression des états internes, et de requêtes pour échanger des informations. Ces mécanismes conduisent à ce que le groupe soit plus efficace. La communication entre les robots dépend de plusieurs critères tels que le degré de sophistication, le type d'informations échangées et le mécanisme de fonctionnement.

4.2.1.1 Communication de haut niveau :

Il s'agit d'une communication qui s'effectue via des mécanismes de protocoles évolués, tels que :

- Etablissement d'une connexion entre deux entités avec des identifiants.
- Existence de protocoles de validation de la réception des messages.

- Echange d'informations de haut niveau citons à titre d'exemple l'état d'achèvement des sous-tâches entreprises par chaque robot (architecture de contrôle ALLIANCE), les plans d'action de chaque robot, la mise aux enchères des tâches qui doivent être exécutées et réalisées par l'allocation dynamique par un groupe de robot (appel d'offre) et l'exécution de la tâche en question par le robot approprié.

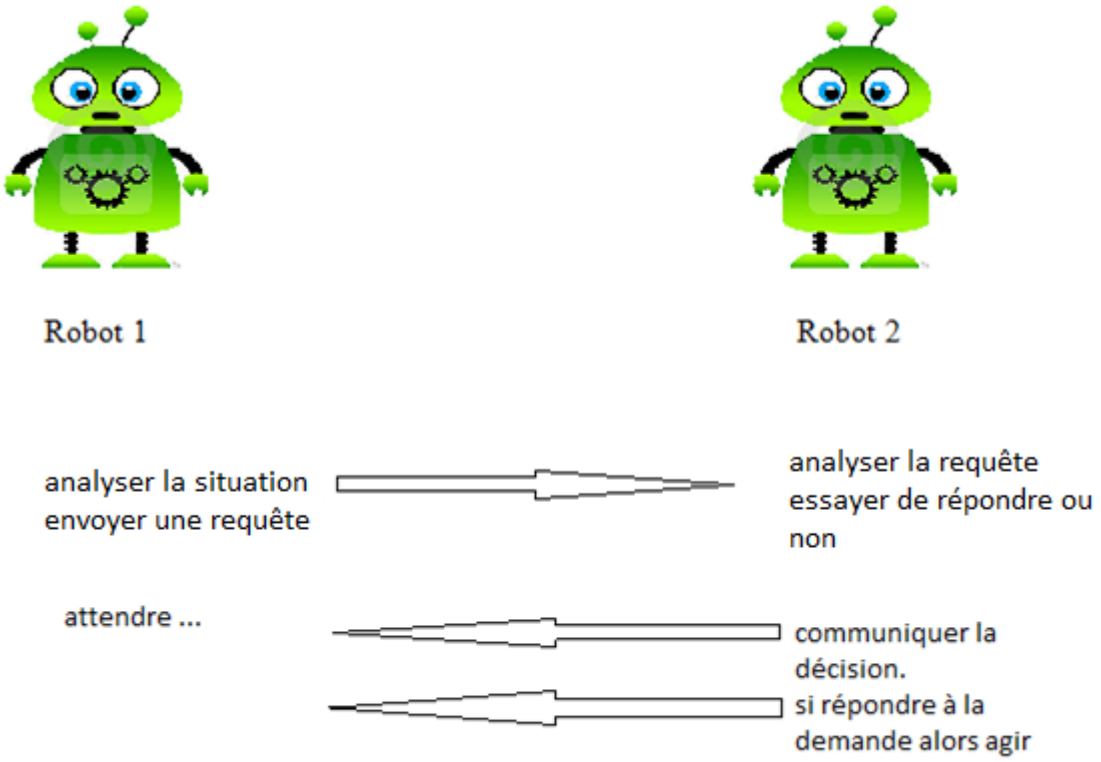

Figure 0-3 : communication de haut niveau

2.4.1.2 Communication de bas niveau :

C'est une communication qui véhicule des informations très simples. Elle ne demande pas un identifiant pour caractériser les émetteurs et les récepteurs de messages qui sont diffusés spontanément. De plus, elle n'exige pas de protocole particulier de négociation entre agents. Ce genre d'interaction entre entités est attribué à des entités réactives, fonctionnant par stimulus-réponse, où le signal reçu par le « robot 2 » est considéré comme un stimulus directement exploitable sans qu'il y ait besoin d'un prétraitement particulier.

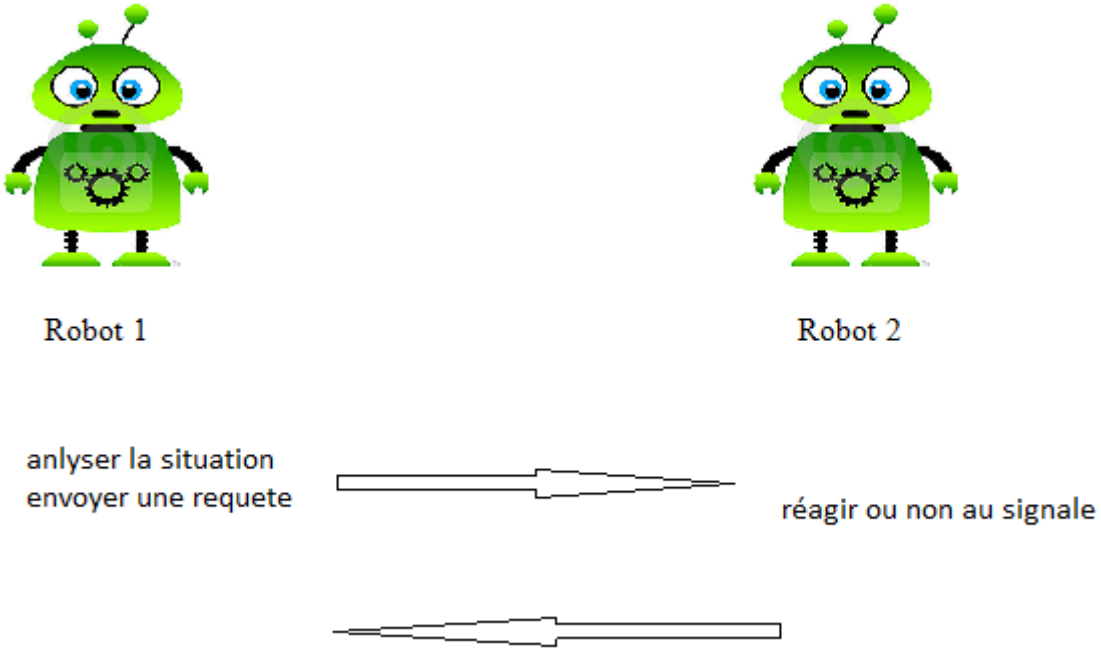

Figure 2-4 : communication de bas niveau

2.4.1.3 Communication indirecte :

C'est une communication à travers l'environnement. Les robots détectent les effets des actions de leur équipe à travers leurs effets sur l'environnement (communication implicite), C'est l'exemple des fourmis qui déposent des phéromones dans l'environnement pour guider leurs congénères vers les sources de nourriture. Cette communication est appelée aussi « stigmergie ».

2.5 Conclusion :

Dans ce chapitre nous avons vu les problèmes des systèmes multi-robots, les approches proposées pour faire communiquer un ensemble de robots, les communications possibles dans un système multi-robots, comme notre travail est focalisé sur la coordination dans les systèmes multi-robots, donc nous serons face à des problèmes de génie logiciel que nous allons aborder dans le prochain chapitre.

Chapitre 3 : Mécanismes de coordination et simulation des scénarios

3.1 Introduction :

Dans ce chapitre nous commençons par décrire la plateforme logicielle et la plateforme matérielle utilisées, en suite nous présentons avec les mécanismes de communications proposés et nous enchainons avec la simulation des différents scénarios proposés.

3.2 Description de la plateforme logicielle :

3.2.1 Python :

Python est un langage portable non seulement sur les différentes variantes d'Unix, mais aussi sur les OS propriétaires, dynamique, extensible, gratuit et qui permet une approche modulaire et orientée objet de la programmation. Python convient aussi bien à des scripts d'une dizaine de lignes qu'à des projets complexes. Un programme Python est souvent de 3 à 5 fois plus court qu'un programme C ou C++ (ou même Java) équivalent, ce qui représente en général un temps de développement de 5 à 10 fois plus court et une facilité de maintenance largement accrue de plusieurs dizaines de milliers de lignes. Python est développé depuis 1989 par Guido van Rossum et de nombreux contributeurs bénévoles. [5]

3.2.2 ROS (Robot Operating System):

C'est un système qui fournit des services proches d'un système d'exploitation pour la robotique de même que les systèmes d'exploitation pour les ordinateurs (abstraction du matériel, gestion de la concurrence, des processus…) mais aussi des fonctionnalités de haut niveau (appels asynchrones, appels synchrones, base centralisée de données, système de paramétrage du robot…). ROS est complètement open source et gratuit pour les utilisateurs, initialement développé par le Laboratoire d'intelligence artificielle de Stanford en 2007, le projet ROS a été adopté par Willow Labs en 2008 et reste à leur charge.

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 (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 continu 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. Au cours de mon stage j'ai utilisé ROS groovy.

3.2.2.1 architecture logicielle :

C'est le réseau peer-to-peer de processus ROS, les concepts de la computation graphique de base de ROS sont : les nœuds, master, paramètre server, les services, les messages et les topics.

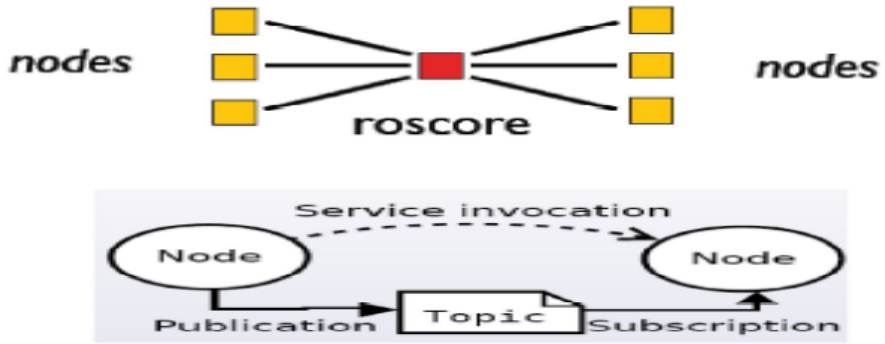

Figure 3-1 mécanisme ROS

Nœuds : c'est un agent de communication avec ROS, c'est une instance d'un exécutable.

Un système de commande du robot comprend de nombreux nœuds, un nœud contrôle les capteurs, un nœud effectue la localisation, un nœud effectue la planification de chemin. Chaque nœud qui se lance se déclare au Master. Les nœuds communique via :

- les topics : c'est un système de transport d'informations basé sur le système d'abonnement / publication (subscribe / publish).

- les services : L'échange d'informations s'effectue d'une manière synchrone requête/ réponse.

Master : est un service de déclaration et d'enregistrement des nœuds qui permet ainsi à des nœuds de se connaître et d'échanger d'informations. Il stocke les sujets et services, les informations d'inscription pour les nœuds ROS

Message : est une structure de donnée composite. Un message est composé d'une combinaison de types primitifs (chaînes de caractères, booléens, entiers, flottants…) et de message (le message est une structure récursive). Les nœuds communiquent entre eux par des messages.

Les services : il y a deux modèles de services :

- Modèle publisher/ subscriber many to many ce modèle est utilisé par les messages

- Modèle requet / reponse ce modèle est utilisé pour les services

3.2.3 Le simulateur Stage :

Le simulateur Stage est un environnement de simulation robots 2D. Stage offre un environnement de simulation de base qui peut être adapté à modéliser un robot. I1 peut être utilisé seul pour simuler les comportements du robot par l'intermédiaire des programmes de contrôle définis par l'utilisateur comme il peut s'interfacer avec Player.

3.3 Plateforme matérielle :

3.3.1Carte Arduino UNO:

Arduino est un circuit imprimé e matériel libre sur lequel se trouve un microcontrôleur qui peut être programmé pour analyser et produire des signaux électriques, de manière à effectuer des tâches très diverses comme le pilotage d'un robot. C'est une plateforme basée sur une interface entrée/ sortie simple.

Arduino peut être utilisé pour construire des objets interactifs indépendants (prototypage rapide), comme elle peut être connectée à un ordinateur pour communiquer avec ses logiciels.

La connexion entre la carte Arduino et l'ordinateur est une connexion série. Elle est établie à travers le port USB, donc il est possible d'envoyer des informations de Arduino à l'ordinateur et de même Arduino peut lire des informations provenant de l'ordinateur.

Figure 03-2 : carte Arduino

La carte Arduino que j'ai utilisé dans mon travail est la Arduino UNO, c'est une carte à microcontrôleur basé sur l'ATmega328 ; elle était utilisée pour communiquer avec le simulateur STAGE et elle a pour rôle de remplacer le robot manipulateur et le capteur du robot mobile dans la simulation.

3.4 Scénario proposé :

Le thème de ce stage est la coordination des robots pour le transport et la manipulation d'objets, son objectif est d'élaborer un scénario d'interaction.

Il s'agit d'un robot manipulateur qui dispose d'un nombre infini d'objets, qu'on veut les transporter à un autre endroit. Pour ce faire, on se sert d'un robot mobile qui s'occupe du déplacement d'objets à l'endroit demandé. La question posée est comment communiquer entre les robots sur ROS.

L'idée générale proposée est inspirée de l'architecture CLIENT/SERVEUR, le robot manipulateur joue le rôle du client qui envoie des requêtes, et le robot mobile joue le rôle du serveur chargé du déplacement de l'objet. Nous allons par la suite présenter les solutions (génie logiciel) proposées pour établir une communication entre les robots.

3.5 Mécanismes de communication :

3.5.1 Les sockets :

Cette technique permet d'établir une communication entre un client est serveur, l'un et l'autre pourront échanger des informations. Le serveur attend l'arrivée de requêtes expédiées par le client par l'intermédiaire d'un port de communication bien déterminé. D'autre part le client tente d'établir la connexion en émettant une requête ; cette requête est un message qui contient dans son entête l'adresse du port de communication destinataires qui est la même que celle du serveur.

Lorsque la connexion est établie avec le serveur, le client lui assigne lui-même l'un de ses propres ports de communication ; à partir de ce moment, un canal privilégié relie le client au serveur, comme si ils sont connectés l'un à l'autre par l'intermédiaire d'un fil (les ports de communication jouent le rôle des deux extrémités de ce fil).

Pour pouvoir utiliser les ports de communications, le client et le serveur font appel à un ensemble de procédures et de fonctions du système d'exploitation, par l'intermédiaire d'objets interfaces qui sont les sockets.

3.5.2 ROS service :

ROS service est une méthode de communication dédié aux outils ; la communication se fait soit par Requête/ Réponse ou Publisher/ Subscriber

Requête / Réponse : elle utilise des messages de structure paire ; une partie requête et une autre partie réponse. Le nœud offre un service sous un nom défini, pour l'utiliser le client envoie un message de demande et il attend la réponse.

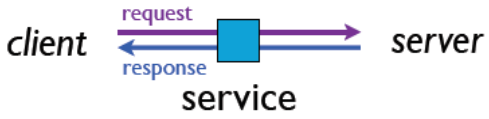

Figure 0-3 Requête/ Réponse

Publisher/ Subscriber : c'est une technique de publication/ souscription des messages dans laquelle l'émetteurs (publisher) publie un message sur un topic et le destinataire (subscricer) souscrit ce message à partir de ce topic.

Figure 3-4 Publisher/ Subscriber

3.5.3 Actionlib :

Il définit une interface standard pour l'interfaçage avec les tâches préemptives, par exemple le déplacement de la base à un endroit visé. Dans les systèmes basés sur ROS, il y a des cas où on veut envoyer une requête à un nœud pour effectuer une tâche, et aussi recevoir une réponse pour cette demande, cela peut être atteint comme on a déjà mentionné via ROS services, si le service prend beaucoup de temps pour l'exécution, l'utilisateur voudrait annuler la requête en cours d'exécution ou obtenir une rétroaction périodique sur la façon dont la requête est en cours, le paquet Actionlib fournit des outils pour créer des serveurs qui exécutent des goals de longue durée qui peuvent être préemptés. Il fournit une interface client pour envoyer des requêtes au serveur [6].

Avec l'Actionlib on peut annuler une tâche, utiliser la préemption et aussi avoir une rétroaction sur la tâche exécutée.

3.5.3.1 Communication entre le client et le serveur :

Le client et le serveur communique entre eux en utilisant « protocol action » ce protocole s'appuie sur des topic ROS afin de transmettre les messages.

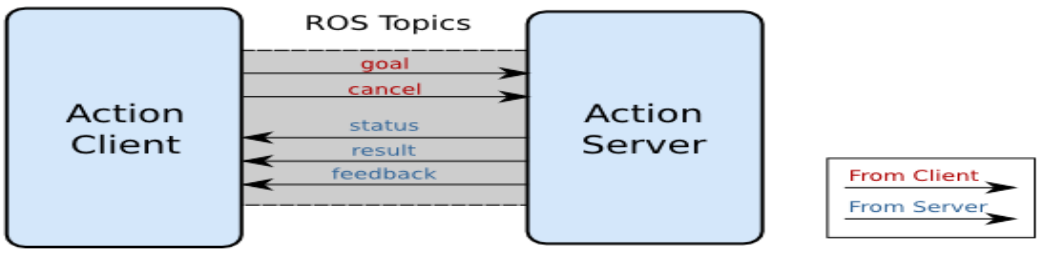

Figure 3-5 : communication entre ActionClient et ActionServer

.

- **Goal** : pour accomplir des tâches en utilisant des actions, on introduit la notion d'un goal qui peut être envoyé à un ActionServer par un ActionClient.

- **Cancel :** il est envoyé à partir du ActionClient au ActionServer, c'est pour permettre d'envoyer des demandes d'annulation au serveur.

- **Feedback :** le Feedback donne des informations sur la tâche qui est encours d'exécution.

- **Status :** c'est pour informer le client sur l'état actuel de goal dans le système.

- **Résultat :** un résultat est envoyé à partir du ActionServer au ActionClient à l'achèvement de l'objectif. Ceci est différent de Feedback car il est envoyé une seule fois.

3.5.3.2 Interaction entre l'ActionClient et l'ActionServer :

Un goal est envoyé par un ActionClient, une fois l'ActionServer reçoit le goal, il crée une machine d'état pour suivre le statut du goal, de l'autre coté l'ActionClient crée une machine d'état secondaire pour suivre l'état du serveur. J'expliquerai l'interaction entre le client et le serveur dans l'annexe A.

3.6 Coordination avec les sockets :

Comme notre idée générale est inspirée de l'architecture CLIENT/SERVEUR, au départ j'ai proposé d'établir une communication entre le client et le serveur via les SOCKETS

J'ai développé des codes pour établir une communication ente un client et un serveur, par exemple, le client envoi une demande et il attend le serveur, si le serveur reçoit la requête il envoie en retour « message reçu », sinon il envoie rien.

J'ai remarqué que cette technique est fiable et sécurisée vu qu'il n'ait pas un risque de perdre l'information ou de se tromper de destinataire. Mais cette technique est redondante pour les outils ROS, ce qui a mené à penser à une nouvelle solution : « ROS service ».

3.7 Coordination avec ROS service :

Les services sont parmi les concepts de computation graph de ROS, d'où j'ai eu l'idée de les exploiter comme mécanisme de communication.

3.7.1 Coordination avec Requête/ Réponse :

Au début j'ai essayé de bénéficier de la méthode Requête/ Réponse vu sa simplicité. En effet, il suffit de créer un package qui offre le service, un client et un serveur. Le client appelle le service par l'envoi d'une requête et il attend la réponse du serveur. Mais cette méthode est inadaptée pour les tâches longues durables dans le temps. Le taux d'échec d'exécution de la tâche dans ce cas sera élevé, par conséquent, je n'ai pas opté pour cette méthode et j'ai proposé de tester la technique Publisher/Subscriber.

3.7.2 Coordination avec Publisher/Subsriber :

L'idée que j'ai proposée est : le robot manipulateur publie sa position sur un topic et le robot mobile reste en écoute, une fois le robot mobile souscrit (reçoit) la position de robot manipulateur, donc il se déplace vers la position reçue.

- Le robot mobile est au point A, il n'a aucun objet sur le dos.
- Le robot manipulateur est au point B
- Le robot manipulateur envoie une demande au robot mobile (en publiant sa position), pour qu'il se déplace vers le point B
- Le robot mobile est en écoute
- Il souscrit ce que le robot manipulateur a publié
- Il se déplace vers le manipulateur
- Si le robot mobile et au point B le robot manipulateur dépose l'objet
- Le robot mobile vérifie si l'objet est bien déposé (en utilisant ses capteurs)
- Le robot mobile déplace l'objet au point C (target)

3.7.2.1 Les étapes de simulation du scénario :

Au départ les coordonnées du robot mobile sont [11.53, 23.21, 0.00, 180.00] , et le robot n'a aucun objet sur le dos, l'état de capture est « False »

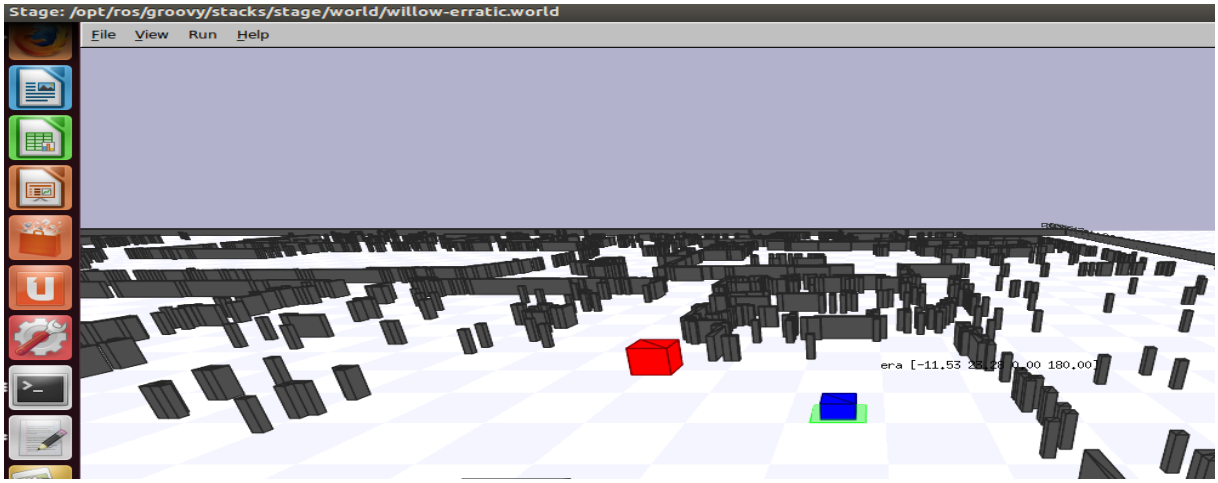

Figure 3-6 : position du robot au départ

```
imene-VPCEH3U1E: ~
imene@imene-VPCEH3U1E:~$ python collaboration.py
the rosdep view is empty: call 'sudo rosdep init' and 'rosdep update'
[INFO] [WallTime: 1379455403.836632] [5780.200000] Pose: (23.27776630358174, 11.
529079737119556, 1.5707963267948966)
(x0,y0)= 23.2777663036 11.5290797371
[INFO] [WallTime: 1379455403.840641] [5780.200000] Pose: (23.27776630358174, 11.
529079737119556, 1.5707963267948966)
(x0,y0)= 23.2777663036 11.5290797371
[INFO] [WallTime: 1379455404.033621] [5780.400000] Pose: (23.27776630358174, 11.
529079737119556, 1.5707963267948966)
(x0,y0)= 23.2777663036 11.5290797371
[INFO] [WallTime: 1379455404.038093] [5780.500000] Pose: (23.27776630358174, 11.
529079737119556, 1.5707963267948966)
(x0,y0)= 23.2777663036 11.5290797371
[INFO] [WallTime: 1379455404.198890] [5780.500000] /talker_arduino: presence obj
 = False
etat de capteur =  False
[INFO] [WallTime: 1379455404.236686] [5780.600000] Pose: (23.27776630358174, 11.
529079737119556, 1.5707963267948966)
(x0,y0)= 23.2777663036 11.5290797371
[INFO] [WallTime: 1379455404.273040] [5780.700000] Pose: (23.27776630358174, 11.
529079737119556, 1.5707963267948966)
(x0,y0)= 23.2777663036 11.5290797371
[INFO] [WallTime: 1379455404.432694] [5780.800000] Pose: (23.27776630358174, 11.
529079737119556, 1.5707963267948966)
(x0,y0)= 23.2777663036 11.5290797371
[INFO] [WallTime: 1379455404.469034] [5780.900000] Pose: (23.27776630358174, 11.
529079737119556, 1.5707963267948966)
(x0,y0)= 23.2777663036 11.5290797371
```

Figure 3-7 : état du capteur au départ

On exécute le programme, le robot fixe publie sa position pour demander au robot mobile de venir vers lui. Ce dernier souscrit ce que le robot fixe a publié (accepte la demande) et il se déplace vers le manipulateur.

```
[INFO] [WallTime: 1379774953.214480] [66528.200000] Pose: (23.209862206546244, 1
1.537471657482387, 1.5707963267948966)
(x0,y0)= 23.2098622065 11.5374716575
[INFO] [WallTime: 1379774953.224235] [66528.300000] pose x: 11.53; pose y: 23.21
[INFO] [WallTime: 1379774953.224679] [66528.300000] /talker_arduino: je recu pos
e x: 11.53; pose y: 23.21
[INFO] [WallTime: 1379774953.352772] [66528.300000] Pose: (23.209862206546244, 1
1.537471657482387, 1.5707963267948966)
```

Figure 3-8 souscription de la publication

une fois le robot mobile arrive au bon endroit (c'est-à-dire il a les mêmes coordonnées que le manipulateur (dans notre exemple [15.04, 23.21,0.00, 180.00]), il s'arrête.

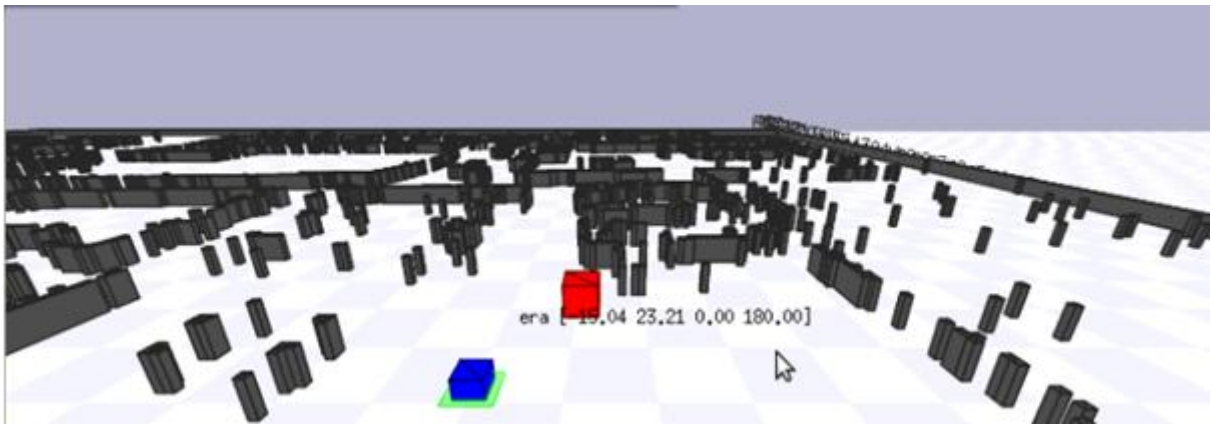

Figure 3-9: le robot mobile au pont B

Le robot fixe dépose l'objet. Pour ce faire on a utilisé la carte Arduino (représente le manipulateur et les capteurs du robot mobile). Si l'objet était bien déposé l'état du capteur passe de « False » à « True ».

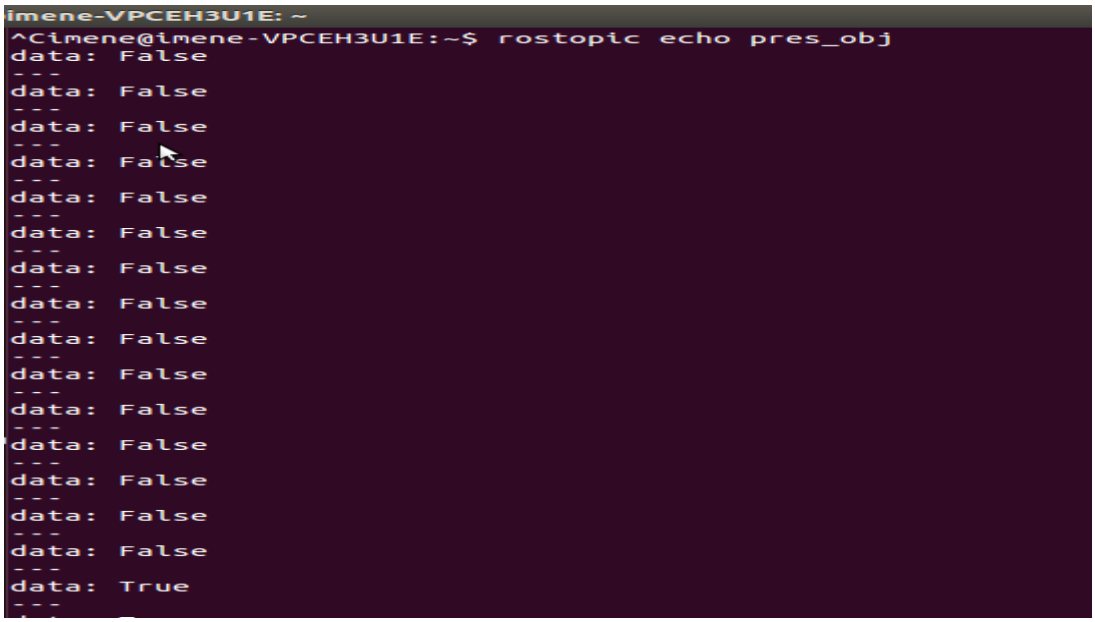

Figure 3-10 : état de capteur après le dépôt de l'objet sur la carte Arduino

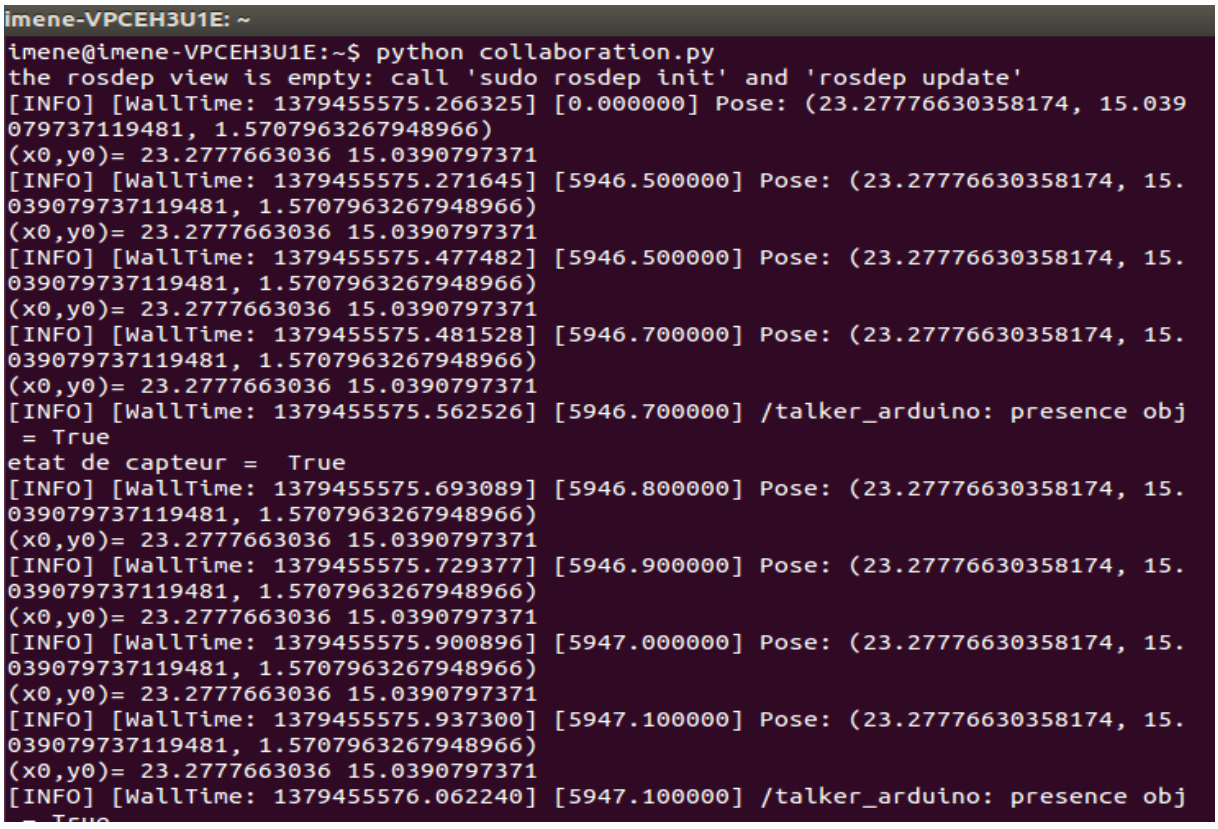

Figure 3-11: état du capteur après le dépôt de l'objet

Après la vérification de la présence de l'objet, le robot mobile se déplace vers le point C où l'objet sera récupéré.

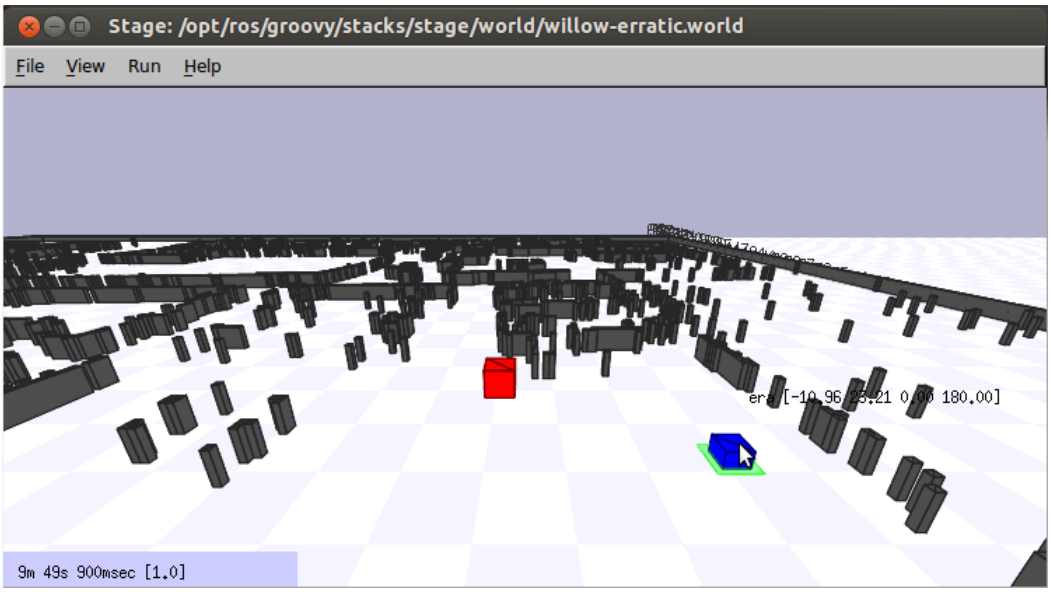

Figure 3-12 : le robot mobile au point C

3.7.2.2 interprétations de résultat de la simulation :

Le robot manipulateur publie sa position en envoyant un message, le robot mobile est en écoute et il a souscrit la position de robot fixe, donc la communication s'est établie entre eux, le robot mobile s'est déplacé vers le robot fixe (point B) , une fois il est arrivé au point B il s'est arrêté, il attend le robot manipulateur pour qu'il dépose l'objet, une fois le manipulateur dépose l'objet l'état de capteur passe de « False » à « True » ce qui signifie que l'objet est sur le dos du robot mobile , par la suite il a déplacé l'objet au point C. toutes les étapes de la simulation se sont déroulées selon le scénario proposé. La tâche globale s'est exécutée avec succès.

Le type de la communication utilisé est une communication de bas niveau, c'est-à-dire la demande envoyée par le robot fixe (le message) est considérée comme un stimulus par le robot mobile.

3.8 Coordination avec Actionlib :

L'idée que j'ai proposée est : l'ActionClient est le robot fixe manipulateur, l'ActionServer est le robot mobile. L'ActionClient envoie un goal et il attend, si l'ActionServer reçoit le goal donc il se déplace vers le manipulateur. Le goal est un message PoseStamped qui contient des informations sur l'endroit où le robot mobile doit se déplacer.

Le feedback est la position actuelle du robot mobile au long du chemin. Le résultat contient la position finale du robot mobile.

3.8.1 Interaction de bas niveau :

- Le robot mobile est au point A, il n'a aucun objet sur le dos.
- Le robot manipulateur est au point B.
- Le robot manipulateur envoie une demande au robot mobile (en envoyant un goal), pour qu'il se déplace vers le point B.
- Le robot manipulateur attend le robot mobile.
- Le robot mobile reçoit le goal et il s'active.
- Il se déplace vers le manipulateur et il envoie SUCCED.
- Si le robot mobile et au point B le robot manipulateur dépose l'objet.
- Le robot mobile vérifie si l'objet est bien déposé (en utilisant ses capteurs).
- Le robot manipulateur envoie un autre goal (les coordonnées du point C).
- Le robot mobile déplace l'objet au point C et à la fin il envoie un résultat qui est sa position finale.

3.8.1.1 Les étapes de la simulation du scénario :

Au départ les coordonnées du robot mobile sont [19.32, 21.17, 0.00, 180.00] , et le robot n'a aucun objet sur le dos, l'état de capture est « False »

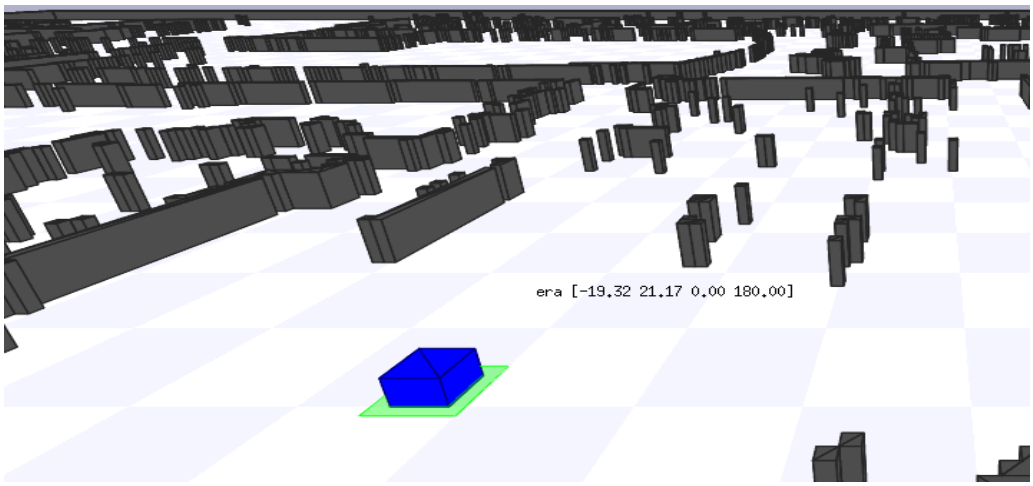

Figure 0-13 : la position du robot au départ

```
imene-VPCEH3U1E: ~
imene@imene-VPCEH3U1E:~$ python server_rob_mobile.py
the rosdep view is empty: call 'sudo rosdep init' and 'rosdep update'
[INFO] [WallTime: 1379447685.367582] [0.000000] Pose: (21.17383585130396, 19.324
219649532676, 1.5707963267948966)
(x0,y0)= 21.1738358513 19.3242196495
[INFO] [WallTime: 1379447685.369420] [9026.500000] Pose: (21.17383585130396, 19.
324219649532676, 1.5707963267948966)
(x0,y0)= 21.1738358513 19.3242196495
[INFO] [WallTime: 1379447685.521463] [9026.500000] /move_base: presence obj = Fa
lse
etat de capteur =  False
[INFO] [WallTime: 1379447685.571729] [9026.600000] Pose: (21.17383585130396, 19.
324219649532676, 1.5707963267948966)
```

Figure 0-14 : état de capture au départ

L'ActionClient envoie le goal (sa position) et il attend le serveur, le serveur (robot mobile) le goal et il s'active et il se déplace vers le robot fixe, si le robot mobile atteint le goal avant l'échéance il envoi SUCCED.

Dans ce cas le robot fixe dépose l'objet. Pour ce faire on a utilisé la carte Arduino (représente le manipulateur et les capteurs du robot mobile). Si l'objet était bien déposé l'état du capteur passe de « False » à « True ».

```
imene-VPCEH3U1E: ~
(x0,y0)= 21.324882515 20.056128193
[INFO] [WallTime: 1379449801.228103] [196.300000] Pose: (21.32488251499349, 20.0
5612819300995, 1.5707963267948966)
(x0,y0)= 21.324882515 20.056128193
[INFO] [WallTime: 1379449801.230511] [196.300000] Pose: (21.32488251499349, 20.0
5612819300995, 1.5707963267948966)
(x0,y0)= 21.324882515 20.056128193
[INFO] [WallTime: 1379449801.283633] [196.400000] /move_base: presence obj = Tru
e
etat de capteur =  True
[INFO] [WallTime: 1379449801.424292] [196.500000] Pose: (21.32488251499349, 20.0
5612819300995, 1.5707963267948966)
(x0,y0)= 21.324882515 20.056128193
[INFO] [WallTime: 1379449801.594153] [196.600000] Pose: (21.32488251499349, 20.0
5612819300995, 1.5707963267948966)
(x0,y0)= 21.324882515 20.056128193
[INFO] [WallTime: 1379449801.596275] [196.700000] Pose: (21.32488251499349, 20.0
5612819300995, 1.5707963267948966)
(x0,y0)= 21.324882515 20.056128193
[INFO] [WallTime: 1379449801.783571] [196.700000] /move_base: presence obj = Tru
e
etat de capteur =  True
```

Figure 0-15 : l'état de capteur après le dépôt de l'objet

L'ActionClient envoie un autre goal qui est la position du pont C (l'endroit où l'objet sera récupéré), le robot mobile accepte le goal et il se déplace au point C ([18.44, 21.32, 0.00, 180.00])

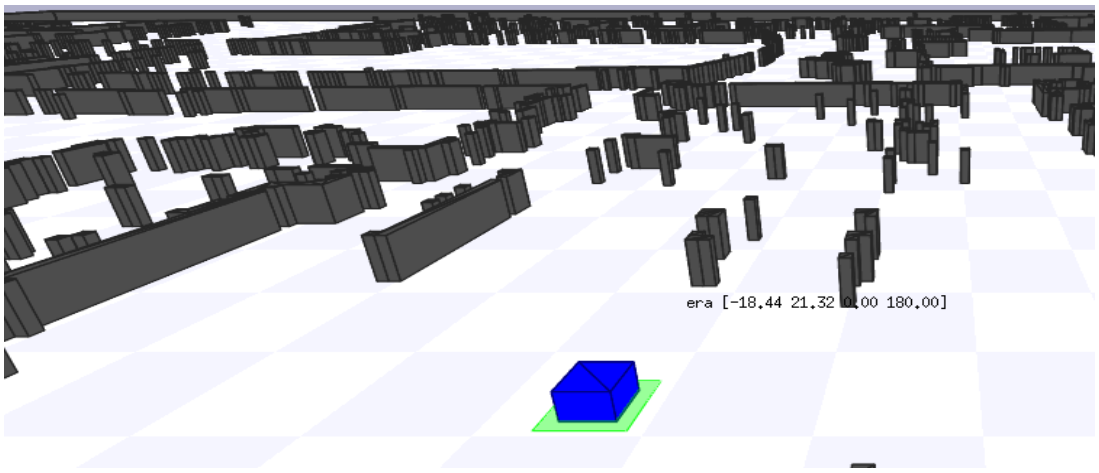

Figure 3-16 robot en point C

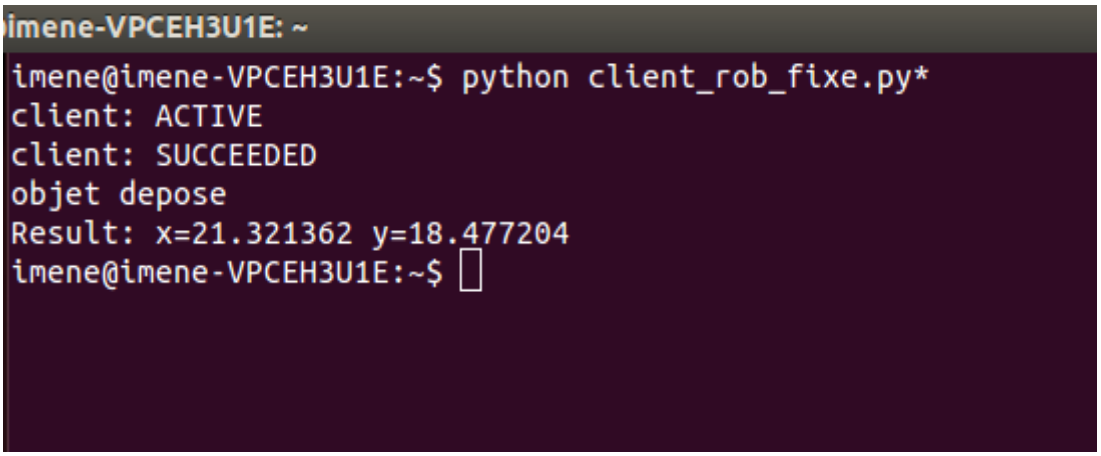

Figure 0-17 : les informations sur le déroulement de la tâche

3.8.1.2 Interprétation de résultat de la simulation :

Le robot mobile s'est activé lorsqu'il a reçu le goal envoyé par le manipulateur, il s'est déplacé vers le point B, une fois il est arrivé au point B il nous renvoie SUCCED, il attend le robot manipulateur pour qu'il dépose l'objet, par la suite il a vérifié si l'objet est bien déposé pour le déplacer au point C et à la fin il a envoyé sa position finale qui se coïncide avec la position de point C. Toutes les étapes de la simulation se sont déroulées selon le scénario proposé, ce qui veut dire que la tâche globale s'est exécutée avec succès.

Nous obtenons le même résultat que le la simulation avec Publisher/Subscriber mais aussi nous obtenons des informations sur le déroulement de la tâche qui est encours d'exécution

3.8.2 Interaction de haut niveau:

Pour montrer le haut niveau d'interaction dans l'Actionlib, j'ai proposé un scénario dans le cas où l'ActionClient définit un temps d'attente pour l'ActionServer, et si l'ActionServer ne répond pas dans le temps défini.

Dans une situation pareille le statut d'ActionServer sera UNKNOWNSTATUS, donc la tâche ne sera jamais exécutée, la solution que j'ai proposée dans le cas où le statut d'ActionServer est UNKNOWNSTATUS est de renvoyer le même goal au ActionServer mais avec un temps de réponse supérieur que le précédant, et comme ça la tâche sera surement exécuté avec succès.

- Le robot mobile est au point A, il n'a aucun objet sur le dos.
- Le robot manipulateur est au point B.
- Le robot manipulateur envoie une demande au robot mobile (en envoyant un goal), pour qu'il se déplace vers le point B.
- Le robot manipulateur attend le robot mobile.
- Le robot mobile reçoit le goal et il s'active.
- Si le robot mobile ne répond pas dans le temps défini par le manipulateur il envoie UNKNOWNSTATUS
- Le robot manipulateur renvoie le même goal avec un temps d'attente plus grand, et il attend
- Le robot mobile reçoit une autre fois le goal et il de déplace vers le point B
- Si le robot mobile et au point B le robot manipulateur dépose l'objet.
- Le robot mobile vérifie si l'objet est bien déposé (en utilisant ses capteurs).
- Le robot manipulateur envoie un autre goal (les coordonnées du point C).
- Le robot mobile déplace l'objet au point C et à la fin il envoie un résultat qui est sa position finale.

3.8.2.1 les étapes de la simulation de scénario :

Les étapes de la simulation sont les mêmes que le scénario précédent sauf que les informations sur la tâche qui est encours d'exécution ne sont pas les mêmes, par exemple le statu de ActionServer n'est pas Succed mais Unknownstatus.

3.8.2.2 interprétations de résultat de la simulation :

Le robot mobile s'est activé lorsqu'il a reçu le goal envoyé par le manipulateur et il a envoyé ACTIVE, puisque le temps d'attente de robot fixe est inferieur de temps de réponse de robot mobile donc il envoie Uknownstatus, puisque le robot fixe a renvoyé le même goal mais avec un temps d'attente supérieur de temps de réponse de robot mobile, le robot s'est déplacé

vers le manipulateur qui a déposé l'objet une fois le robot mobile est au point B, donc l'état de capteur est passé de « False » à « True », en suite le robot mobile a déplacé l'objet au point C.

Ce qui signifie que la tâche globale s'est exécutée avec succès.

3.8.3 Scénario proposé dans le cas où il y a plusieurs robots mobiles :

Les scénarios proposés précédemment s'agissent d'une situation où notre système multi-robots est composé d'un seul robot fixe et un seul robot mobile. Le scénario que j'ai proposé dans cette partie concerne un système multi-robots qui est composé d'un seul robot manipulateur, et plusieurs robots mobiles. Dans cette situation c'est le robot manipulateur qui gère la coordination entre les robots.

Le robot manipulateur envoie le goal au « robot mobile_1 » et il attend un certain temps, si le « robot mobile_1 » ne répond dans le temps défini par l'ActionClient il envoie le même goal au « robot_mobile_2 » et il attend certain temps et si ce dernier ne répond pas dans le temps défini par l'ActionClient il renvoie le goal aux autres robots mobiles, jusqu'au un robot mobile accepte le goal.

3.9 La navigation :

Pour le déplacement du robot mobile dans l'environnement j'ai proposé d'utiliser une fonction « move_to », cette fonction publie des commandes de vitesse suivant l'axe des X, et une fonction « callback_odometry » pour récupérer la position actuelle du robot mobile le long de son trajet.

Comme la fonction « move_to » publie les commandes de vitesse que suivant l'axe des X, donc le déplacement suivant l'axe des Y n'est pas possible, par conséquent les simulations précédents sont valables que lorsque les coordonnées Y du robot manipulateur et le point C sont les mêmes que les coordonnées Y de robot mobile, et dans un endroit où il n'y a pas d'obstacle. Pour rendre les scénarios valables dans toutes les situations, nous avons proposé d'utiliser la navigation.

La navigation est l'une des fonctions les plus importantes pour le robot ; elle permet d'éviter les obstacles, et de planifier des chemins globaux dans l'environnement.

Dans ROS, il existe une STACK NAVIGATION qui automatise et coordonne ces différentes étapes.

La version de ROS que j'ai utilisée au début de mon stage, elle n'offre pas ce package, donc je n'ai pas réussi à utiliser la navigation dès le début, à la fin du stage, j'ai eu une nouvelle version de ROS qui offre ce package. J'ai commencé à essayer de simuler les scénarios précédents en utilisant la navigation, mais mes essais n'ont pas abouti à des résultats.

3.10 Comparaison des différentes solutions proposées :

- Les sockets : sont fiables et sécurisées mais sont redondants pour les outils ROS
- ROS service :
    - Requête/réponse : ce n'est pas convenable pour les tâches durables dans le temps
    - Publisher/Subscriber : nous avons pu établir une communication entre le client et le serveur, et la tâche s'est exécutée avec succès. Cette technique est simple à implémentée mais elle est recommandée que pour une communication de bas niveau
- Actionlib : ce mécanisme est le plus robuste puisqu'il permet une interaction de haut niveau, annuler ou préempter la tâche qui est encours d'exécution, renvoyer le goal au cas d'un échec ; mais il est un peu compliqué pour l'implémentation.

3.11 Conclusion :

Dans ce chapitre nous avons présenté des différents scénarios d'interaction entre le robot mobile et le robot manipulateur et leurs simulations, par la suite nous avons analysé le résultat de chaque simulation.

Conclusion générale :

Dans le cadre de ce projet, j'ai élaboré plusieurs scénarios qui permettent à un système multi-robot de coordonner pour transporter un objet d'un endroit A à un entre endroit.

L'interaction dans les scénarios proposés est : soit une interaction de bas niveau, ou bien une interaction de haut niveau.

Dans le cas d'une interaction de bas niveau j'ai utilisé deux technique de communication, Publisher/ Subscriber et l'Actionlib, dans le cas d'une interaction de haut niveau par exemple pour l'annulation ou la préemption de la tâche j'ai utilisé l'Actionlib.

La simulation du scénario pour une interaction de bas niveau avec les deux techniques a donné un bon résultat, la communication entre les robots s'est établie, et la tâche s'est exécutée avec succès. Mais avec l'Actionlib le résultat est plus précis parce que on peut avoir des informations sur la tâche encours, par exemple en peux avoir les différents statuts de robots mobile lors de l'exécution de la tâche.

Dans le cas d'une interaction de haut niveau j'ai utilisé l'Actionlib qui offre la possibilité d'annuler ou de préempter une tâche.

La simulation du scénario pour une interaction de haut niveau a donné des résultats acceptables, même dans le cas où le robot mobile répond après l'échéance, la tâche s'est exécuté avec succès grâce à l'Actionlib qui permet de renvoyer un goal avec un temps d'attente supérieur pour éviter l'échec de la tâche.

Dans le cadre de mon travail j'ai proposé différents scénarios d'interaction, leurs simulations sur le simulateur STAGE a présenté de bons résultats. Je suggère très vivement les prochains intervenants sur ce projet d'implémenter ces scénarios sur des robots réels, afin de confirmer l'exactitude des résultats obtenus et les comparés avec des travaux précédents pour vérifier la robustesse et l'efficacité de ce travail.

Référence :

[1] : T. Arai, E. Pagello, and L.E. Parker. Editorial : Advances in multi-robot systems.

IEEE Transactions on robotics and automation, 18(5) :655–661, 2002.

[2] : L Adouane. These : Architecture des Contrôle comportementales et Réactives pour la Coopération d'un Groupe de Robots Mobiles.

[3] : Multiple Mobile Robot Systèms

[4] : Matthis Radestock , S Eisenbach. Cordination in Evolving Systems. Departments of computing Imperial College of Science, Technology and Medicine

[5]: G.Swinnen. Apprendre à programmer avec Python. Copyright (c) 2000-2008 Gérard Swinnen

[6]: I Lukebohle, R Philippsen, V Pradeep, E Marder-Eppstein, S Wachsumuth. Generic middleweresupport for coordinating robot software components : The Task-State-Pattern.

Journal of Software Engineering for Robotics

Lien utiles:

www.ros.org

http://wiki.ros.org/actionlib/DetailedDescription

http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29

http://web.engr.oregonstate.edu/~smartw/me539/

http://inforef.be/swi/python.htm

ANNEXE A

Les interactions entre l'ActionServer et l'ActionClient

- **Description de ActionServer**
  L'ActionServer crée une machine d'état

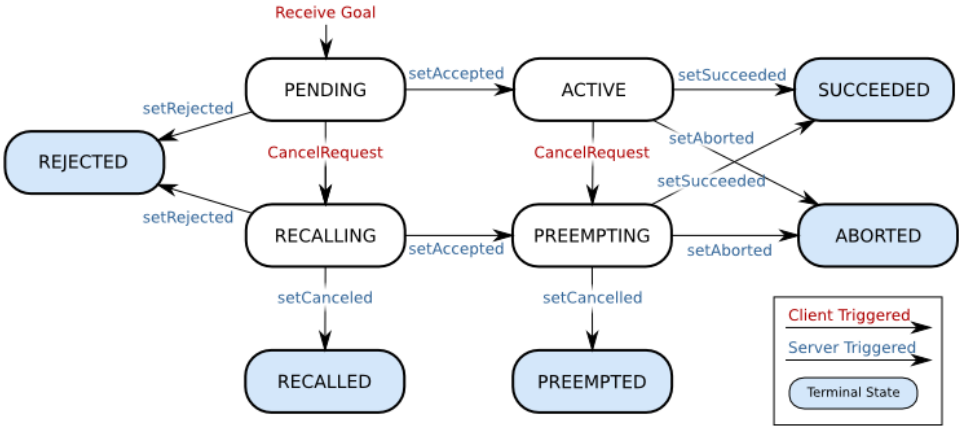

Figure 4-1 état de machine de ActionServer

l'ActionClient envoie le goal, l'ActionServer reçoit le goal et il attend : soit il décide de ne pas traiter la demande et rejeter la demande donc sont état finale est REJECTED, ou il accepte la demande, alors il s'active. L'ActionServer est activé deux cas sont possible soit il nous prévient que la demande est traitée avec succès est sont état final est SUCCEEDE, soit il nous prévient que le goal a rencontré une erreur, donc son état final est ABORTED.

Dans le cas où l'ActionServer reçoit le goal et il attend, l'ActionClient peut informer l'ActionServer que le goal envoyé est annulé, l'état d' ActionServer est RECALLING, trois cas sont possible, soit il décide de ne pas traiter la demande et son état final est REJECTED, soit le goal n'est plus en cours de traitement et son stat final est RECALLED, ou il accepte la demande est son état transitoire est PREEMPTING ce qui veut dire qu'une demande d'annulation est envoyée du ActionClient mais l'ActionServer n'a pas confirmé l'annulation du goal ; trois états finaux sont possible : SUCCED dans le cas où la demande est traitée avec

succès, ABORTED dans le cas où le goal a rencontré une erreur, ou PREEMPTED dans le cas où le goal n'est plus en cours de traitement, en raison d'une demande d'annulation.

- **Description du client :**

L'ActionClient envoie le goal et il suit l'état du server suivant ce schéma :

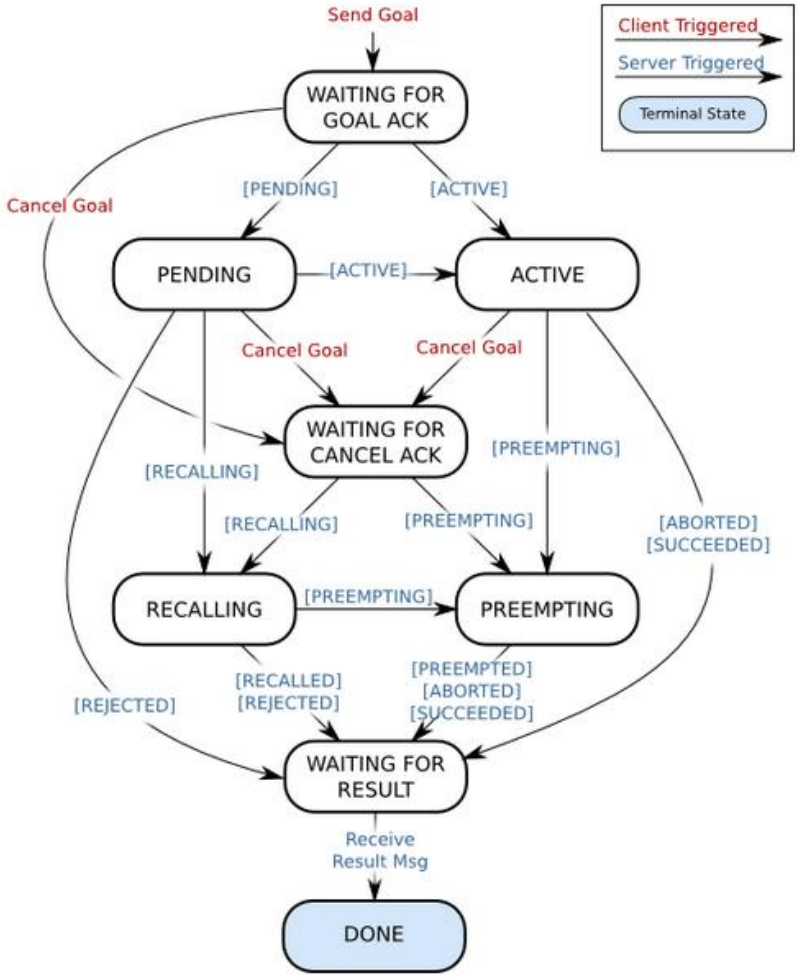

Figure 0-2 machine d'état de ActionClient

ANNEXE B

Code python

server_rob_mobile.py

```
#! /usr/bin/env python
# see http://docs.ros.org/hydro/api/actionlib/html
# code by, Imene.BOUYOUCEF Sept. 2013.

import roslib; roslib.load_manifest('roslissi_code')
import rospy
from geometry_msgs.msg import Twist
from std_msgs.msg import Bool
from nav_msgs.msg import Odometry
import actionlib
import roslissi_code.msg
from tf.transformations import euler_from_quaternion
from geometry_msgs.msg import PoseStamped

class SrvMobileRobot(object):
        # create messages that are used to publish feedback/result
        _feedback = roslissi_code.msg.MoveBaseFeedback()#send_object.msg.MoveBaseAction()
        _result   = roslissi_code.msg.MoveBaseResult() #send_object.msg.MovebaseAction()

        def __init__(self, name):
                self.data_sub = rospy.Subscriber("pres_obj", Bool, self.callback)
```

```
                self.odom_sub = rospy.Subscriber('base_pose_ground_truth', Odometry,
self.odom_callback)

                self._action_name = name


                _as = actionlib.SimpleActionServer(self._action_name,
roslissi_code.msg.MoveBaseAction, execute_cb=self.execute_cb, auto_start=False)

                _as.start()

                self._as = _as

                self._feedback.base_position = PoseStamped()

         self.tol_x = 1 # tolerance

                self.pub = rospy.Publisher('cmd_vel', Twist)

                self.pres_obj

        def callback(self, data):

                rospy.loginfo(rospy.get_name() + ": presence obj = %s" % data.data)

                self.pres_obj= data.data

                print "etat de capteur = ", self.pres_obj


        def odom_callback(self, odom):

        angles = euler_from_quaternion([odom.pose.pose.orientation.x,

                        odom.pose.pose.orientation.y,

                        odom.pose.pose.orientation.z,

                        odom.pose.pose.orientation.w])

        self.pose = (odom.pose.pose.position.x, odom.pose.pose.position.y,

             angles[2])

        rospy.loginfo('Pose: {0}'.format(self.pose))

        self.x0= (odom.pose.pose.position.x)

                self.y0= (odom.pose.pose.position.y)

        print "(x0,y0)=",self.x0, self.y0
```

```
def execute_cb(self, goal):

        r = rospy.Rate(10)
        success = True
        # check that preempt has not been requested by the client
        if self._as.is_preempt_requested():
    rospy.loginfo('%s: Preempted' % self._action_name)
    self._as.set_preempted()
    success = False



        else:
    success = True
    # read target position
    x_target = goal.target_pose.pose.position.x
    y_target = goal.target_pose.pose.position.y
            delta_x = 10.
            eps = 1.0
            rospy.sleep(1)
            while abs(delta_x)>self.tol_x:
                        # publish motor command
                        twist=Twist ()
                        x=  self.x0#   (position actuelle)
                        y=  self.y0#   (position actuelle)
                        # compute delta_x
                        if delta_x > 0:
```

```
                    twist.linear.x = 0.2


                if delta_x < 0:
                    twist.linear.x=-0.2


            self.pub.publish(twist)
                delta_x = (x_target-y)
                #publish feedba
                self._feedback.base_position.pose.position.x = self.x0
                self._feedback.base_position.pose.position.y = self.y0
                self._as.publish_feedback(self._feedback)
                rospy.loginfo('action name: %s;  position x= %f, y= %f' %
(self._action_name,
self._feedback.base_position.pose.position.x,self._feedback.base_position.pose.position.y ))

    if success:

        self._result.base_position.pose.position.x = self.x0
        self._result.base_position.pose.position.y = self.y0
        rospy.loginfo('%s: Succeeded' % self._action_name)
        self._as.set_succeeded(self._result)
    else :
        self._result.base_position.pose.position.x = self.x0
        self._result.base_position.pose.position.y = self.y0

  rospy.loginfo('%s: Aborted' % self._action_name)
```

```
                self._as.set_aborted(self._result)


if __name__ == '__main__':
  rospy.init_node('move_base')
  SrvMobileRobot(rospy.get_name())
  rospy.spin()
```

client_rob_fixe.py

```
#! /usr/bin/env python

# coded by Imene.BOUYOUCEF, Sept. 13.

import roslib;
import rospy
import actionlib
from actionlib_msgs.msg import *
from geometry_msgs.msg import PoseStamped
import roslissi_code.msg
from std_msgs.msg import UInt16

pub = rospy.Publisher('put_obj', UInt16)
```

```
def put_obj():
    if not rospy.is_shutdown():
            pub.publish(data=1)



def ClientRobFixe():
    exec_wait_time = 50

    client = actionlib.SimpleActionClient('move_base',
roslissi_code.msg.MoveBaseAction)#send_object.msg.MoveBaseGoal
    client.wait_for_server()

    # Creates a goal to send to the action server.
    p = PoseStamped()
    p.pose.position.x = 21.0
    p.pose.position.y = 1.0

    goal = roslissi_code.msg.MoveBaseGoal(target_pose=p)

    # Sends the goal to the action server.
    client.send_goal(goal)
    client.wait_for_result(rospy.Duration(exec_wait_time) )
    s =client.get_state()

    if s==GoalStatus.ACTIVE:
       print "client: ACTIVE"
       client.wait_for_result(rospy.Duration(exec_wait_time) )
```

```
    s =client.get_state()
    if s==GoalStatus.SUCCEEDED:
        print "client: SUCCEEDED"
            l= put_obj()
            print "objet depose"
            rospy.sleep(2)
            p.pose.position.x= 17.5
        p.pose.position.y= 1.0
        goal_1 = roslissi_code.msg.MoveBaseGoal(target_pose=p)
            client.send_goal(goal_1)
            client.wait_for_result(rospy.Duration(exec_wait_time) )

    elif s==GoalStatus.ABORTED:
        print "client: ABORTED"
    else:
        print "client: GoalStatus unknown"
    return client.get_result()

if __name__ == '__main__':
    try:
        # Initializes a rospy node so that the SimpleActionClient can
        # publish and subscribe over ROS.
        rospy.init_node('ClientFixedRobot')
        result = ClientRobFixe()
            print "Result: x=%f y=%f "%(result.base_position.pose.position.x,
result.base_position.pose.position.y)
    except rospy.ROSInterruptException:
        print "program interrupted before completion"
```

Résumé :

Les systèmes multi-robots ont fait une grande révolution dans plusieurs domaines grâce aux leurs avantages par rapport aux systèmes mono-robot, tels que la rapidité, la robustesse, la flexibilité. Parmi les tâches réaliser par un système multi-robot est le transport d'objets, pour ce faire, les entités de ce système doivent coordonner et communiquer entre elles. La communication peut être soit une communication de haut niveau ou une communication de bas niveau.

Les techniques utilisées pour coordonner un ensemble de robots sont :

- Les Sockets : redondant pour les outils ROS.
- ROS services
  Requête/ Réponse : inadaptée pour les tâche durables dans le temps.

  Publisher/ Subscriber : permet une interaction de bas niveau.
- Actionlib :
  Permet une interaction de haut niveau.

Abstract:

Multi-robot systems have been a big revolution in several areas due to their advantages over monorobot systems, such as speed, strength, flexibility. Among the tasks performed by a multi-robot system is transporting objects to do this, the entities of the system must coordinate and communicate. Communication can be either a high level of communication or low-level communication.

The techniques used to coordinate a group of robots:

- The Sockets: redundant for ROS tools.
- ROS server:

  Request/ reponse: unsuitable for sustainable task in time.

  Publisher/ Subscriber: Allows low-level interaction
- Actionlib: Allows high-level interaction