

# **MANUEL D'UTLISATION DU ROBOT PEOPLEBOT**

*par Pierre MARQUESTAUT  
Professeur d'Informatique et Télématique*

*mis à jour le 20 juin 2006*

# LICENSES

Cette création est mise à disposition selon le Contrat Paternité-ShareAlike 2.5 disponible en ligne <http://creativecommons.org/licenses/by-sa/2.5/deed.fr> ou par courrier postal à Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

PeopleBot(tm), MobileSim(tm), Mapper3Basic(tm) and MobileEyes(tm) sont des marques de la société MobileRobots Inc.

La librairie ARIA est fournie avec le robot PeopleBot sous licence GNU GPL.

This work is licensed under the Creative Commons Attribution-ShareAlike 2.5 License. To view a copy of this license, visit [http://creativecommons.org/licenses/by-sa/2.5/deed.en\\_GB](http://creativecommons.org/licenses/by-sa/2.5/deed.en_GB) or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

PeopleBot(tm), MobileSim(tm), Mapper3Basic(tm) and MobileEyes(tm) are trademarks of MobileRobots Inc.

The library ARIA comes with the robot PeopleBot under GNU GPL.

# Table des matières

LICENSES.....	2
INTRODUCTION.....	5
I. GENERALITES.....	6
1) QU'EST-CE QU'UN ROBOT ?.....	6
2) DOMAINES D'APPLICATION.....	6
3) CLASSIFICATIONS.....	6
II. LE ROBOT PEOPLEBOT(tm).....	8
1) DESCRIPTION.....	8
2) SYSTEME D'EXPLOITATION.....	8
3) COMMUNICATION .....	9
A) Configuration Série.....	9
B) Adresse IP.....	9
C) Sélection du médium.....	9
D) Point d'Accès.....	10
E) Configuration.....	10
4) INTERFACAGES AVEC LE ROBOT .....	11
A) MODE Console.....	11
B) MODE Telnet.....	11
C) MODE VNC.....	11
5) TELECOMMANDES.....	13
A) Joystick.....	13
B) L'Application DEMO.....	14
C) MobileEyes(tm).....	15
III. ARCOS.....	17
1) Qu'est-ce qu'ARCOS ? .....	17
A) Une Architecture Client/Serveur.....	17
B) Interface.....	17
2) Protocole de communication.....	17
A) Paquet de Commande.....	18
B) SIP (Server Information Packet).....	19
C) Déroulement d'une Connexion.....	19
D) QoS : Qualités de service.....	19
IV. SIMULATION.....	21
1) MAPPER3BASIC(tm).....	21
2) MOBILESIM(tm).....	22
V. APPLICATION CLIENTE.....	25
1) ARIA.....	25
2) CYCLE DE FONCTIONNEMENT.....	26
A) Principe.....	26
B) Déclenchement.....	26
C) Modes de Fonctionnement.....	26
3) DEPLACEMENTS.....	27
A) Décomposition des mouvements.....	27
B) Suivi de lignes de couleur.....	27
C) Positions.....	27
4) ACTIONS.....	27
A) Principe .....	27
B) Types d'actions.....	28
C) Action désirée.....	28

D) Asservissements.....	28
E) Actions prédéfinies.....	29
5) TACHES UTILISATEURS.....	29
A) Principe.....	29
B) Intégration dans le cycle.....	29
C) Automate à Etats Finis.....	30
6) COMMUNICATION TCP/IP.....	33
A) Pont TCP/Série.....	33
B) Choix du protocole de Transport.....	33
7) COMPILATION.....	34
A) Compilation sous LINUX.....	34
B) Compilation sous MICROSOFT WINDOWS .....	35
C) Compilation de la bibliothèque dynamique.....	36
8) JAVA.....	36
A) JNI.....	36
B) Limitation.....	36
C) Compilation.....	36
9) CREATION D'UNE IHM.....	37
A) Les pages Web dynamiques.....	37
B) Un interface JAVA.....	37
VI. AUDIO/VIDEO.....	39
1) CONVERSION VIDEO.....	39
2) MANIPULATION DE LA CAMERA.....	39
3) RECONNAISSANCE D'IMAGE.....	39
A) Phase d'entraînement.....	40
B) Serveur ACTS.....	40
C) Intégration dans une application cliente.....	41
4) TRAITEMENT DE L'IMAGE.....	41
A) Exploitation des ressources.....	41
B) Librairies.....	43
5) SYNTHESE VOCALE.....	43
6) RECONNAISSANCE VOCALE.....	43
7) TRANSMISSION DU SON ET DE L'IMAGE.....	43
VII. RESOLUTION DE PROBLEMES.....	45
1) Erreur au démarrage de MobileSim sous Windows XP.....	45
2) Erreur à l'exécution d'un programme utilisant la librairie Aria sous Linux.....	45
3) Erreur de connexion d'une application à MobileSim.....	45
4) Erreur à la compilation avec Aria sous Linux.....	46
5) Erreur de liens à la compilation de la librairie sous Visual C++ .....	47
6) Problème de DLL à l'exécution d'un programme.....	48
LEXIQUE.....	49
REFERENCES LOGICIELLES.....	50
DOCUMENTATION.....	50
ANNEXE 1 : Tableau des codes de commande.....	51
ANNEXE 2 : Tableau des commandes implémentées dans MobileSim 0.2.....	53

# INTRODUCTION

Destiné aux différents utilisateurs du robot - que ce soient professeurs, ou élèves dans le cadre d'un projet, le but de ce manuel est d'expliquer le fonctionnement du robot ainsi que les différentes façons de l'exploiter.

Ce manuel ne se prétend pas être exhaustif : il s'agit d'un condensé de différents manuels, notes relatifs au robot et à ses dépendance. Pour des informations plus précises, je vous conseille de vous reporter directement vers ces documents dont vous trouverez la liste à la section DOCUMENTATION de ce manuel.

En dehors du contenu de ces manuels, j'ai rajouté également des éléments découverts lors de différentes expérimentations, recherches ou conversations avec le support technique des sociétés ActivMedia Robotics/MobileRobots, Inc. (constructeur) et Robosoft (diffuseur en France).

Enfin, je ferai parfois référence à certains TP que j'ai rédigés et qui illustrent de façon expérimentale certains de mes propos.

# **I. GENERALITES**

## **1) QU'EST-CE QU'UN ROBOT ?**

L'origine du mot robot est issue du tchèque "robota" qui signifie travail forcé. Le terme de robotique est apparu en 1942 dans l'oeuvre de l'écrivain Isaac ASIMOV.

On peut donner comme définition actuelle qu'un robot est une machine programmable, mobile ou non, qui imite des actions d'une créature intelligente.

## **2) DOMAINES D'APPLICATION**

Utilisés dans les années 60 pour des opérations hasardeuses, on distingue de nos jours trois grands domaines d'application de la robotique :

- Production et Fabrication industrielle (soudage, assemblage, détournage, peinture...) ;
- Exploration dans des endroits dangereux ou inaccessibles (exploration spatiale, sous-marine...) ;
- Services aux humains et aux équipement (sauvegarde des personnes, nettoyage maintenance, surveillance, transport...).

## **3) CLASSIFICATIONS**

Dans la robotique, on peut distinguer - entre autres - les robots industriels, composés d'un bras articulé muni d'un effecteur, et le robots mobiles qui peuvent se déplacer en autonomie dans un environnement.

Cependant, certains organismes, comme la JIRA (Association Japonaise des Robots Industriels) ou l'AFRI (Association Française de Robotique Industrielle), ont établi des classifications plus précises, basées sur la spécificité fonctionnelle des robots.

La classification élaborée par la JIRA est la suivante :

- Classe 1 : TELEMANIPULATEURS
  - Bras commandé par un opérateur humain.
- Classe 2 : MANIPULATEURS AVEC SEQUENCE FIXE
  - Contrôle automatique, mais difficilement programmable.
- Classe 3 : MANIPULATEURS AVEC SEQUENCE VARIABLE
  - Contrôle automatique, reprogrammé mécaniquement
- Classe 4 : ROBOTS « PLAY-BACK »
  - Séquences qui sont exécutées à l'origine sous la supervision d'êtres humains, mémorisées puis rappelées pour être rejouées.
- Classe 5 : ROBOTS AVEC CONTROLEUR NUMERIQUE
  - Les positions des séquences sont contrôlées par des données numériques.
- Classe 6 : ROBOTS INTELLIGENTS
  - Le robot peut réagir à son environnement et à des modifications arrivant durant l'exécution.

Pour information, voici également la classification de l'AFRI :

- Classe A : TELEMANIPULATEURS
  - Manipulateur maître/esclave
- Classe B : MANIPULATEURS AUTOMATIQUES
  - Manipulateurs automatiques avec séquences fixes
  - Manipulateurs automatiques avec séquences variables (Machines à commandes numériques)
- Classe C : ROBOTS PROGRAMMABLES
  - 1ère génération de robots
- Classe D : ROBOTS INTELLIGENTS
  - 2ème génération de robots, munis de système de vision
  - 3ème génération de robots, avec intelligence artificielle.

Trois caractéristiques différencient le robot intelligent d'un jouet :

- Il peut suivre ses propres mouvements ;
- Il peut « ressentir » son environnement ;

Il peut prendre des décisions sur l'action à suivre, basées sur les dernières informations.

## II. LE ROBOT PEOPLEBOT<sup>tm</sup>

### 1) DESCRIPTION

Le robot PeopleBot<sup>tm</sup>, développé par la société ActivMedia, a été spécialement conçu pour interagir avec son environnement, et plus particulier avec l'homme.

Il peut ainsi :

- parler,
- obéir à des ordres vocaux,
- se déplacer de façon autonome,
- éviter des obstacles,
- reconnaître des objets,
- les attraper,
- etc...

### 2) SYSTEME D'EXPLOITATION

Le robot dispose système d'exploitation embarqué. Il s'agit de Red Hat Linux version 7.3 (Noyau 2.4.18-5 sur un i686).

Cet OS dispose entre autre d'un environnement graphique X-window - sous interface Gnome - indispensable notamment pour l'utilisation du logiciel ACTS.





Le compte par défaut est :

login : "guest"

passwd : ""

Le compte root n'a également par défaut aucun mot de passe.

### **3) COMMUNICATION**

Le robot dispose de 3 médiums de communication possibles :

- liaison série RS232,
- liaison Ethernet RJ45,
- liaison Ethernet WiFi (Radio Ethernet).

#### **A) Configuration Série**

La configuration du port série est la suivante :

- vitesse : 9600 bauds ;
- taille des données : 8 bits ;
- parité : aucune ;
- bit des stop : 1 ;
- contrôle de flux : aucun.

#### **B) Adresse IP**

L'adresse IP peut soit être fixe (configurée via le système d'exploitation), soit être dynamiquement configurée grâce au protocole DHCP .

Le robot dispose d'une adresse IP pour chacune de ces interfaces Ethernet.

Liaison filaire : 192.168.109.222

Liaison sans fil : 192.168.109.250

Les adresses IP peuvent toutefois être vérifiées par la commande `/sbin/ipconfig`.

#### **C) Sélection du médium**

Il est nécessaire de configurer le système d'exploitation pour déterminer lequel du RJ45 ou du WiFi doit être initialisé au démarrage du système.

Pour ce faire, on dispose dans le répertoire `/etc/sysconfig/network-scripts` les fichiers `ifcfg-eth0` et `ifcfg-eth1` , correspondant respectivement à la liaison RJ45 et à la liaison WiFi. Dans ces fichiers, nous trouvons la propriété « onboot » : si cette propriété est mise à « yes », la liaison correspondante est initialisée au démarrage du système. Ainsi pour passer d'une liaison Ethernet à l'autre, il faut modifier ces fichiers (« onboot » étant donc à « yes » pour la liaison voulue) puis redémarrer le système.

Attention : un et un seul de ces fichiers doit avoir sa propriété mise à « yes ».

### **D) Point d'Accès**

Le robot est livré avec un point d'accès de marque Cisco. Il dispose en outre d'un serveur Web et d'un serveur DHCP embarqué. Disposant d'une prise RJ45, ce point d'accès peut être connecté à un ordinateur ou directement à un réseau local.

Accessible via n'importe quel navigateur, le serveur Web embarqué permet de configurer le point d'accès et notamment les paramètres de la liaison WIFI.

Son adresse IP actuelle est 192.168.109.224.

Si l'on ignore cette adresse, il existe un moyen simple de la retrouver en utilisant le serveur DHCP embarqué :

- Brancher le point d'accès à un ordinateur ;
- Lancer le logiciel Ethereal pour intercepter les paquets qui circulent entre l'ordinateur et le point d'accès éteint ;
- Configurer l'ordinateur en client DHCP, autrement pour qu'il récupère dynamiquement son adresse IP ;
- Allumer le point d'accès ;
- Récupérer une trame DHCP dans lequel le point d'accès informe l'ordinateur de sa nouvelle adresse IP : l'adresse IP du point d'accès s'y trouve aussi.

### **E) Configuration**

Si on souhaite que le robot soit en liaison avec le point d'accès fourni ou tout autre point d'accès WIFI, il est nécessaire de configurer l'interface WIFI en mode "Managed". Dans le cas contraire, c'est-à-dire avec un réseau de deux machines, elle doit alors être en mode "Ad-Hoc".

Il existe 2 moyens d'effectuer cette configuration,

- soit provisoirement par l'instruction `/usr/local/sbin/iwconfig` ;
- soit en modifiant la configuration par défaut.

Cette configuration par défaut se trouve dans le fichier `/etc/pcmcia/wireless.opts` et est lancée au démarrage de la machine. On trouve dans ce fichier tous les éléments de configuration.

Exemple : Configuration pour le point d'accès CISCO:

```
ESSID="Wireless Network"  
MODE="Managed"  
RATE="auto"
```

Si le réseau sans fil sur lequel on souhaite connecter la machine dispose d'une clé WEP, il faut alors décommenter dans le fichier la ligne KEY et renseigner ce champ.

Si votre clé WEP *ma\_clé* est en hexadécimal, taper **KEY="*ma\_clé*"**.

Si elle est en ASCII, rajouter "s:" : **KEY="*s:ma\_clé*"**.

## **4) INTERFACAGES AVEC LE ROBOT**

### **A) MODE Console**

Il est possible d'accéder au système d'exploitation embarqué sur le robot en y connectant un moniteur, une souris et un clavier.

On accède alors au système d'exploitation en mode « lignes de commande ». On peut ensuite basculer vers un environnement graphique Xwindow par la commande *startx*.

L'inconvénient de ce mode est évidemment de ne pouvoir configurer le robot à distance.

### **B) MODE Telnet**

Au travers de la liaison série ou d'une liaison Ethernet, on peut accéder au système d'exploitation du robot grâce au protocole Telnet, géré par une fenêtre de commandes DOS, l'Hyperterminal ou tout autres applications.

L'interface avec le système ne se fera ici que par lignes de commandes. Il n'est donc pas possible de bénéficier de l'environnement graphique Xwindow.

### **C) MODE VNC**

Le programme VNC (Virtual Network Computing) est un logiciel de contrôle à distance, qui permet de visualiser ou de commander (selon les droits accordés) un bureau distant (ici, le système d'exploitation embarqué).

Le programme se compose naturellement d'une application serveur qui s'exécute sur l'OS du robot, et d'une application cliente qui s'exécute à partir duquel on souhaite contrôler le robot.

Ce logiciel permet donc d'avoir la maîtrise à distance et en mode graphique de l'OS embarqué du robot. Il est cependant à noter que des ralentissements au niveau de l'affichage peuvent survenir par cette méthode.

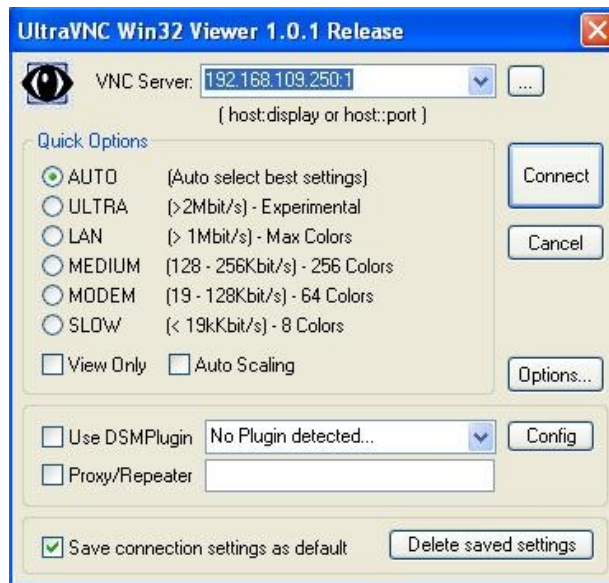
Au niveau du robot, *vncserver* - la partie serveur de l'application - se trouve dans */etc/sysconfig* et est théoriquement lancé au démarrage grâce à *inetd*. Il est à noter que l'application se lance dans un mode d'affichage donné, référencé par un nombre (par défaut : 1).

Pour se connecter depuis un poste distant (sous Windows ou Linux), on peut utiliser l'application *vncviewer*, soit – et de préférence – l'application *UltraVNCviewer*.

Ce freeware fonctionne sur la même technologie que VNC et se trouve même être compatible avec *vncserver*. Il est cependant plus efficace et offre davantage de fonctionnalités. Couplé avec *UltraVNCserver*, il peut également transférer des fichiers avec le PC distant.

Au démarrage de l'application client (VNCviewer ou UltraVNCviewer), il est nécessaire de renseigner l'adresse IP du robot ainsi que le numéro de l'affichage.

Exemple : 192.168.109.250:1



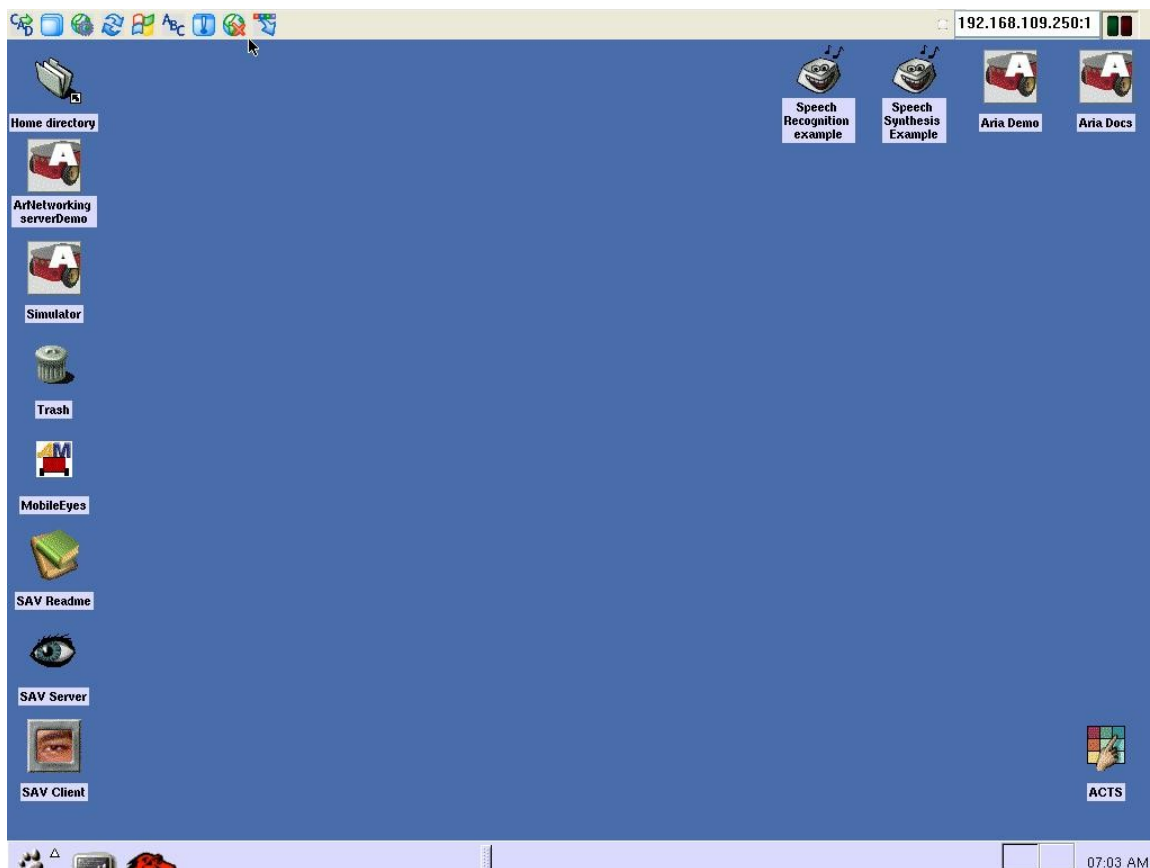
Ensuite, pour des raisons de sécurité, l'utilisation du logiciel impose la saisie d'un mot de passe (ici le mot de passe est : vnc\_robot).

Pour modifier ce mot de passe, il faut exécuter, côté serveur, l'application en ligne de commande vncpasswd.



A l'ouverture de la connexion avec le serveur, l'environnement X-window s'ouvre automatiquement en mode root.

La barre d'outils insérée par UltraVNC permet entre autres choses d'envoyer des commandes clavier (ex : CTRL-V, CTRL-N...) vers l'ordinateur serveur.



## 5) **TELECOMMANDES**

Le robot PeopleBot offre la possibilité d'être utilisé comme un robot de classe A (ou 1, selon la classification), c'est-à-dire directement commandé par un opérateur humain. (Pour une commande plus avancée, se reporter au Chapitre V : Application cliente.)

### **A) Joystick**

Fourni avec le robot et branché sur une prise dédiée, le joystick permet de manipuler le robot avec des commandes rudimentaires :

- Marche Avant
- Marche Arrière
- Arrêt
- Rotation
- Réglage de la vitesse

Outre le peu de commandes, le principal inconvénient du joystick est la connection filaire, qui oblige l'opérateur à rester constamment à proximité du robot (entre 1 et 2 mètres).

Il est à noter qu'il est possible d'intégrer les instructions du joystick à l'intérieur d'une application cliente (Cf. Chapitre V) au travers de les classes ArJoyHandler et ArActionJoydrive.

## **B) L'Application DEMO**

L'application open-source demo.exe est fournie avec la bibliothèque ARIA dans le répertoire *Aria 2.4.0/bin* et ses codes dans le répertoire *Aria 2.4.0/examples*.

Application en ligne de commandes (sous X-term ou sous fenêtre DOS), elle offre une utilisation complète du robot et de ses périphériques (pince, caméra...). Elle propose également l'affichage en temps réel des relevées faites par le robot (position, vitesse, état de la pince...).

```
D:\PROJET ROBOT\ARIA 2.4-0\bin>demo.exe
You may press escape to exit
Connecting to simulator through tcp.

Syncing 0
Syncing 1
Syncing 2
Connected to robot.
Name: MobileSim
Type: Pioneer
Subtype: peoplebot-sh
Loaded robot parameters from peoplebot-sh.p
ArACTS_1_2::openPort: Open failed: Connection refused.

You can do these actions with these keys:

quit: escape
help: 'h' or 'H' or '?' or '/'

You can switch to other modes with these keys:
          tcm2 mode: 'm' or 'M'
    command mode: 'd' or 'D'
          acts mode: 'a' or 'A'
           io mode: 'i' or 'I'
position mode: 'p' or 'P'
      bumps mode: 'b' or 'B'
       sonar mode: 's' or 'S'
      camera mode: 'c' or 'C'
    gripper mode: 'g' or 'G'
    wander mode: 'w' or 'W'
unguarded teleop mode: 'u' or 'U'
      teleop mode: 't' or 'T'
      laser mode: 'l' or 'L'

You are in 'teleop' mode currently.

Teleop mode will drive under your joystick or keyboard control.
It will not allow you to drive into obstacles it can see,
though if you are presistent you may be able to run into something.
For joystick, hold in the trigger button and then move the joystick to drive.
For keyboard control these are the keys and their actions:
  up arrow: speed up if forward or no motion, slow down if going backwards
  down arrow: slow down if going forwards, speed up if backward or no motion
  left arrow: turn left
  right arrow: turn right
  space bar: stop
transVel    rotVel      x          y          th          volts
  443        0         1056        0         0.0         13.0
```

L'avantage de cette application est de pouvoir manipuler le robot à distance, le laissant totalement libre de ses mouvements.

On peut soit lancer cette application depuis le système d'exploitation du robot, soit depuis un poste distant.

Dans le premier cas, il est donc possible de se connecter à distance à l'OS (Cf paragraphe 4 : interfaçages avec le robot) et d'y lancer l'application. Il est à noter qu'une simple connexion Telnet suffit.

Dans le second cas, l'application tentera de se connecter au robot via le port série COM1. Pour un port différent, il faudra utiliser l'option « -rp ». De plus, pour se connecter via le protocole TCP/IP, il faudra alors utiliser l'option « -rh », le port étant fixé par défaut à 8101.

Exemples :

```
>demo.exe -rp COM2
```

```
>demo.exe -rh 192.168.109.222
```

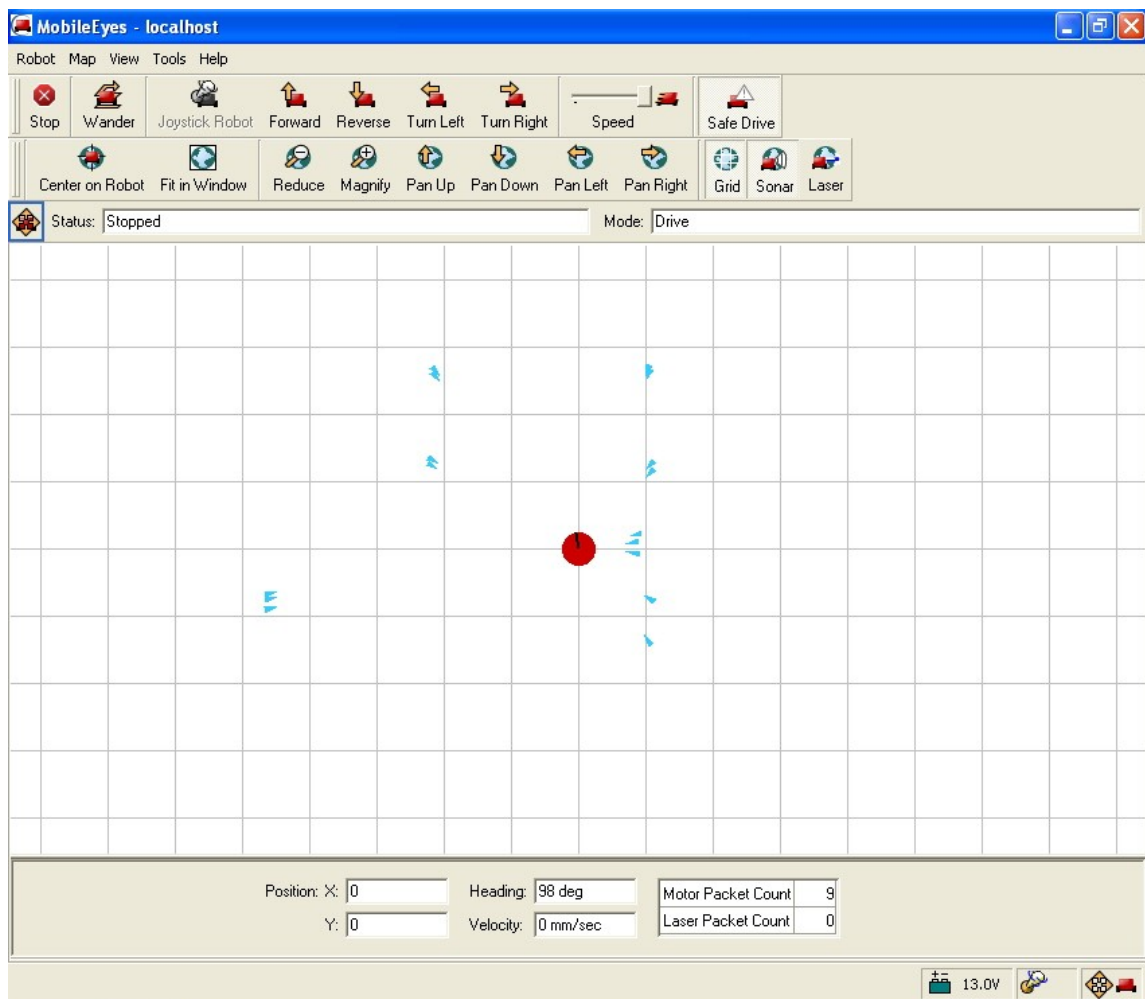
Cette application a également la possibilité de fonctionner avec un simulateur (CF Chapitre IV). Elle tentera d'ailleurs au lancement de s'y connecter par défaut.

### **C) MobileEyes<sup>(tm)</sup>**

Contrairement à l'application demo.exe, MobileEyes<sup>(tm)</sup> dispose d'une interface graphique (GUI en anglais).

Les commandes proposées sont moins nombreuses que dans l'application demo.exe (pas de gestion des périphériques...), mais il est ici possible de suivre graphiquement l'avancée du robot.

Tout comme dans l'application précédente, on manipuler le robot à distance, utiliser le robot ou un simulateur, lancer l'application depuis le robot ou un poste distant. Il est à noter que l'application nécessite obligatoirement d'un environnement graphique et que l'environnement X-Window du robot devra par conséquent être lancé. L'utilisation de cette application avec Telnet est ainsi impossible.



Sur la capture d'écran ci-dessus, on distingue le robot, représenté par un disque rouge avec un trait noir pour indiquer le devant de l'appareil. Les traits bleus tout autour représentent les valeurs des rangées de sonar. Ici on peut voir la présence d'un obstacle sur la droite du robot, là où les traits bleus sont les plus rapprochés.



### III. ARCOS

#### 1) *Qu'est-ce qu'ARCOS ?*

##### **A) Une Architecture Client/Serveur**

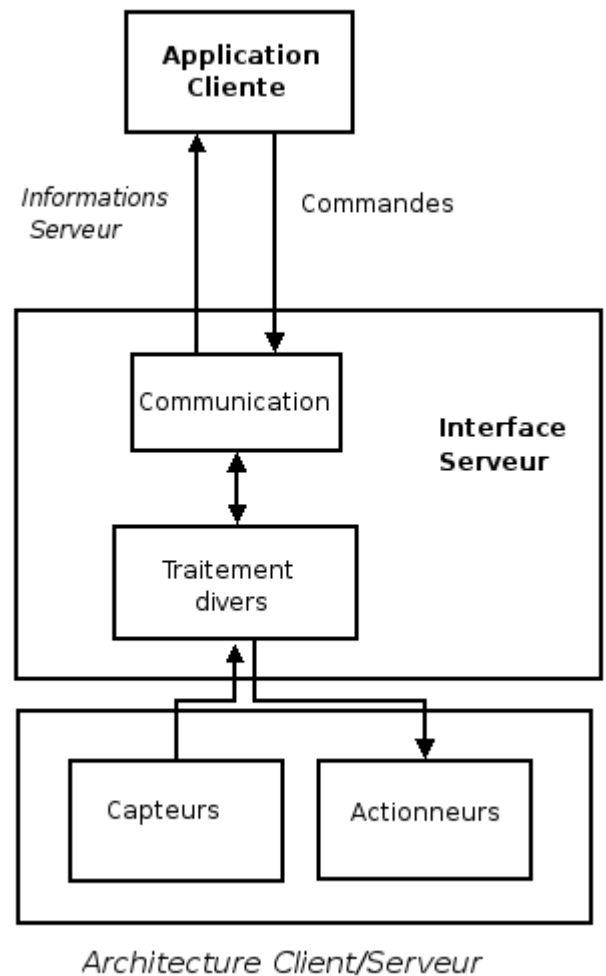
Le robot repose sur une architecture client/serveur. En outre, il dispose d'un contrôleur embarqué, fonctionnant en tant que serveur et dédié à la manipulation des informations de bas niveau (capteurs et actionneurs).

##### **B) Interface**

ARCOS (ActivMedia Robot Control et Operation Software) est une interface client/serveur, initialisée au démarrage du robot, au travers de laquelle il est possible de contrôler le robot et de collecter des informations, ce qui comprend les mouvements des roues, les valeurs retournées par les sonars, etc...

On peut accéder directement à ARCOS à partir de l'OS embarqué grâce à l'application ARCOScf : cette application en ligne de commande permet d'effectuer des réglages et de contrôler le robot.

Les paramètres d'ARCOS (temps de cycle, durée du watchdog, paramètres du PID...) sont contenues dans la mémoire Flash du robot et peuvent être modifiés par l'envoi d'une commande. (par ARCOScf ou toute autre application cliente.)



#### 2) *Protocole de communication*

Afin de transmettre les informations nécessaires au contrôle du robot, il a été développé un protocole de communication propriétaire. Ce protocole se compose de 2 types de paquet :

- les paquets de commandes de l'application cliente vers le robot,
- les informations serveurs (ou SIP) du robot vers l'application cliente.

## A) Paquet de Commande

Entete	Longueur	Commande	Type	Argument	Checksum
--------	----------	----------	------	----------	----------

*Format des paquets de commande*

Entête (2 octets) : Il s'agit invariablement des caractères 0xFA et 0xFB.

Longueur (1 octet) : Nombre d'octets du paquet, checksum compris, sans compter l'entête ni le champ longueur.

Commande (1 octet) : Code de la commande de 0 à 255 (*Cf Annexe 1*)

Type (1 octet) : Il s'agit du type de l'argument de la commande :

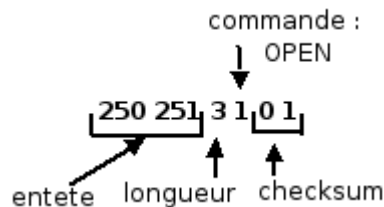
- Entier positif : 0x3B
- Entier négatif ou absolu : 0x1B
- String : 0x2B

Argument (n octets) : Argument de la commande. (Si la taille de l'argument est supérieur à 1 octet, l'argument est alors précédé de sa taille.)

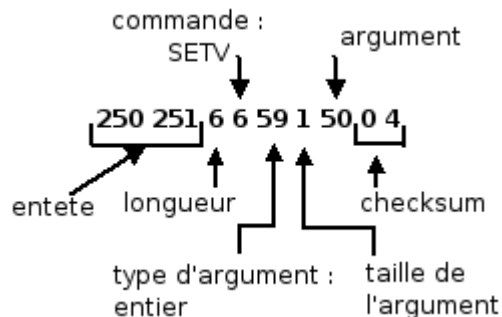
Checksum (2 octets) : Champ de vérification de l'intégrité du paquet. Il est à remarquer que l'octet de poids faible précède ici l'octet de poids fort.

### Exemples :

- Paquet qui permet de l'ouverture des serveurs du robot :



- Paquet qui fixe la vitesse du robot à 50 mm/s :



## **B) SIP (Server Information Packet)**

Une fois la connexion établie (Cf Chapitre suivant), ARCOS transmet automatiquement et régulièrement un SIP standard au travers du médium de transmission.

Le format des ces paquets est similaire à celui des paquets de commandes.

## **C) Déroulement d'une Connexion**

### **Connexion au robot.**

Comme nous l'avons déjà vu, le robot repose sur une architecture client/serveur : une application cliente doit par conséquent se connecter au robot serveur. Pour se faire, l'application doit transmettre dans l'ordre trois paquets de synchronisation (référéncés par SYNC0, SYNC1, SYNC2) au robot : si elle reçoit en retour des paquets en écho, alors la connexion est établie.

Bien que la connexion soit établie, il reste indispensable de commander l'ouverture des serveurs du robot (sonars, moteurs...) par la transmission de la commande OPEN.

### **Pulsation.**

Comme indiqué précédemment, le robot transmet à l'application cliente un paquet SIP tous les 100ms. En retour, côté robot, un "watchdog"(en français, "chien de garde") s'attend à recevoir de façon régulière un paquet de commande de la part de l'application. S'il ne reçoit pas de paquet durant un laps de temps prédéfini, alors la connexion client/serveur s'interrompt et le robot s'arrête.

De ce fait, quand aucune commande ne doit être envoyée au robot, l'application cliente doit envoyer des paquets de pulsation, qui ne contient aucune commande autre que la commande dédiée PULSE, à seule fin de réinitialiser le "watchdog".

### **Déconnexion**

L'envoi par le client d'un paquet avec la commande CLOSE permet à la fois d'arrêter le robot et de clôturer la communication. Le robot se met alors en attente d'un nouveau client.

## **D) QoS : Qualités de service.**

Il se peut que lors de la transmission d'un paquet (qu'il s'agisse d'un SIP ou d'un paquet de commande) on trouve une erreur au niveau du checksum : dans tous les cas, le paquet est ignoré et non redemandé.

En effet, pour des raisons de temps réel, cela demande moins de temps de rejeter complètement un paquet et d'attendre le suivant, plutôt que de redemander à l'expéditeur un paquet mal transmis. On ne travaille alors qu'avec les informations les plus récentes. Un système avec ce comportement est qualifié d'application temps réel "soft".

De la même manière, aucun accusé de réception (mise à part l'écho pour la connexion) n'est envoyé. Cette absence de QoS a comme autre avantage d'éviter la surcharge du médium de communication.

Ainsi, Si un paquet de commande est perdu ou rejeté, alors c'est au client qui a émis cette commande de s'apercevoir grâce aux SIP suivants que sa commande n'a pas été prise en compte et qu'il doit donc la retransmettre.

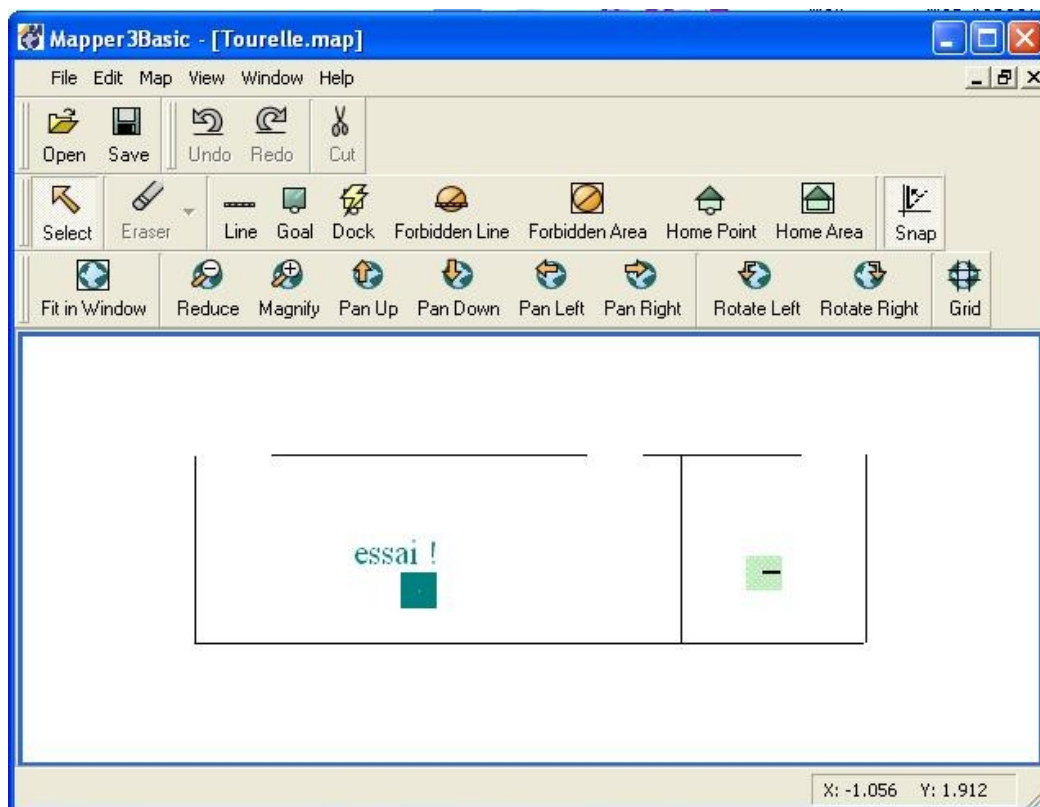
## IV. SIMULATION

Il existe un logiciel de simulation qui prend en charge les différents modèles de robots de la société. On peut ainsi faire manipuler plusieurs élèves en même temps sans avoir besoin de disposer de plusieurs robots.

Cette simulation a la possibilité d'utiliser des plans préalablement créés dans Mapper3Basic, ce qui permet d'apprécier les trajectoires du robot dans un environnement « accidenté ».

### 1) **MAPPER3BASIC<sup>(tm)</sup>**

Le logiciel Mapper3Basic<sup>(tm)</sup> permet de créer des cartes en définissant des murs, des obstacles ainsi qu'un point de départ pour le robot.



Chargée par le logiciel de simulation, les différents éléments de la carte seront pris en compte dans les déplacements du robot.

Les cartes, générées en .map, peuvent également être exploitées dans une application client au travers de la classe ArMap de la librairie ARIA.

## 2) **MOBILESIM<sup>(tm)</sup>**

MobileSim<sup>(tm)</sup> est un logiciel de simulation créé spécialement pour tester une application cliente avant de l'utiliser directement sur le robot.

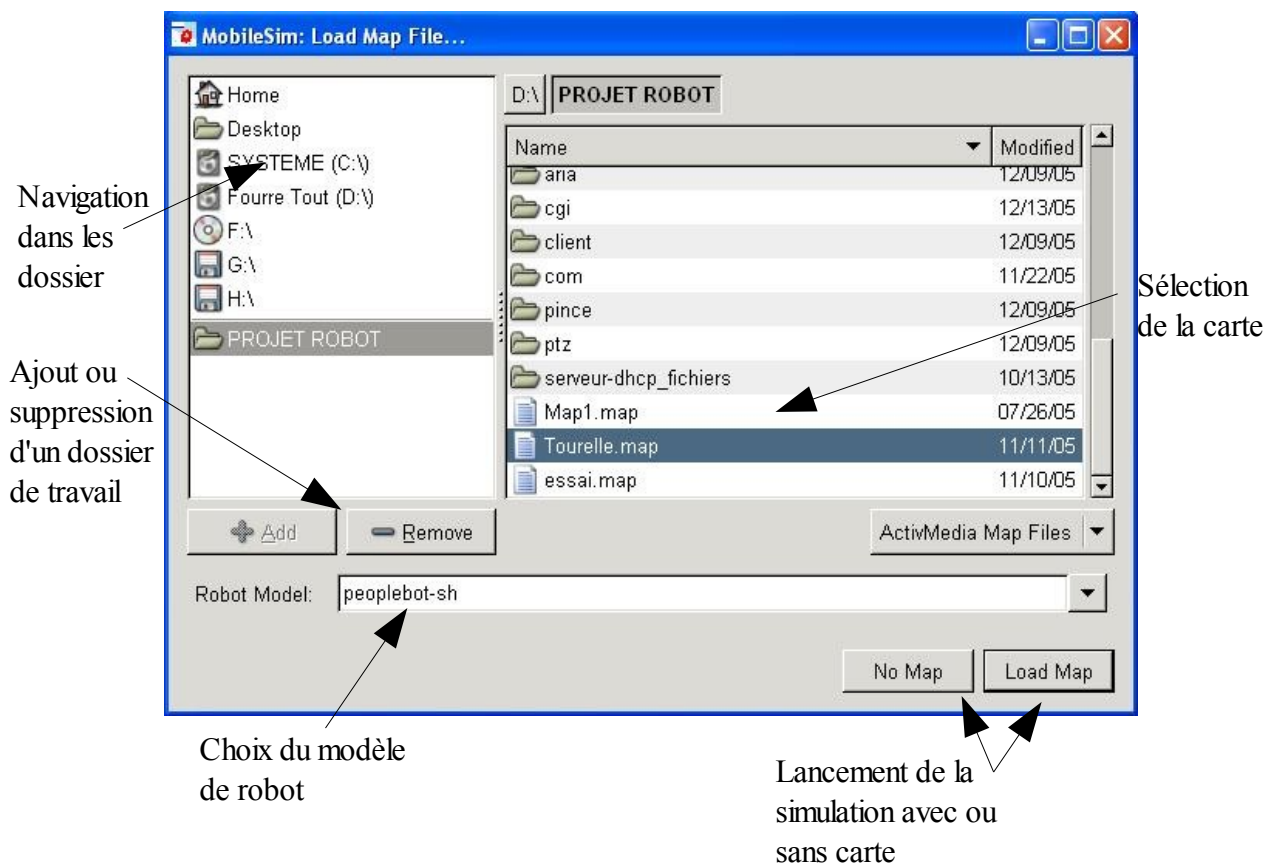
A l'ouverture du simulateur, il nous est proposé différents modèles de robot. Dans notre cas, on choisira le modèle "peoplebot-sh". (En cas de problème avec ce modèle, le modèle p3dx, proche de celui de peoplebot, peut éventuellement être utilisé.)

Sur la même fenêtre, il est également possible - mais pas indispensable - de sélectionner une carte. Ceci permettra de représenter le robot à l'intérieur d'un plan comprenant murs et obstacles.

Pour ce faire, il faut choisir la carte désirée puis cliquer sur "Load Map".

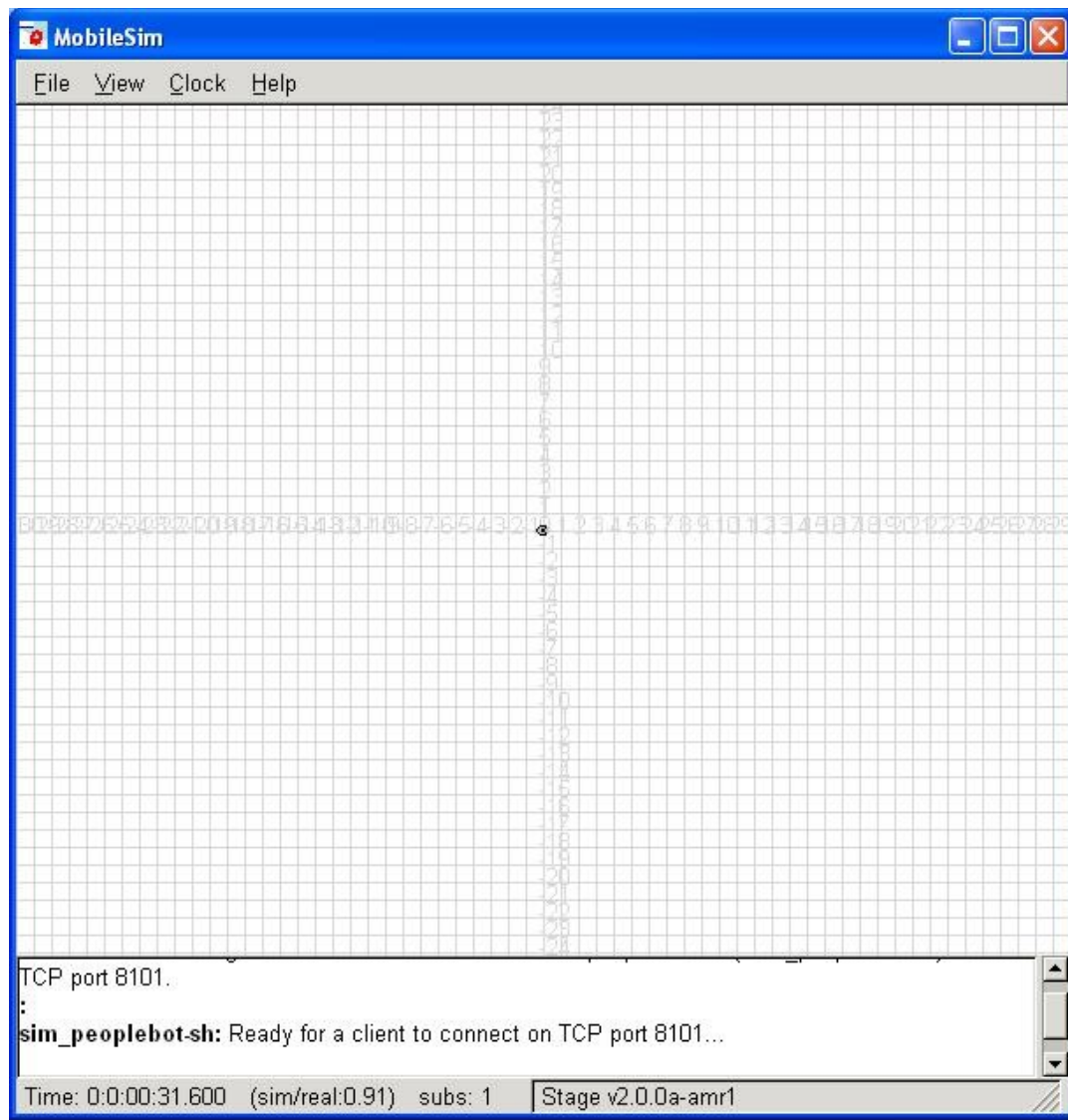
Pour une simulation sans carte, cliquer sur "No Map".

Dans les 2 cas, la simulation se lance.



Une fois la simulation lancée, une application cliente peut alors s'y connecter par liaison TCP/IP sur le port 8101. (Pour simuler un communication par liaison série, il faudra par conséquent utiliser un pont entre les protocoles RS233 et TCP/IP.)

Par ailleurs, la méthode *openSimple()* de la méthode *ArTcpConnection* tente par défaut de se connecter un simulateur situé sur la même machine.



Le robot simulé à l'écran se comporte exactement comme le robot réel, et dispose des même capteurs, comme notamment les sonars. Cependant, la plupart de périphériques (caméra, pince...) - à l'instar de certaines commandes - ne sont pas encore implémentés.

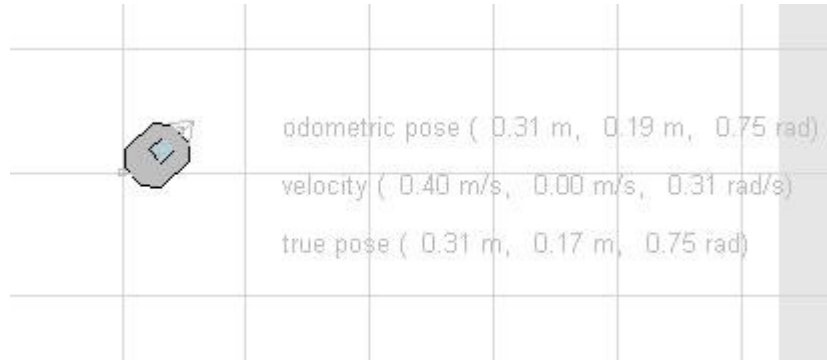
Si le simulateur reçoit une commande non implémentée ou un paquet destiné à un périphérique inconnu, un message d'alerte s'affiche.



Il est possible d'afficher temps réel les informations de position du robot.

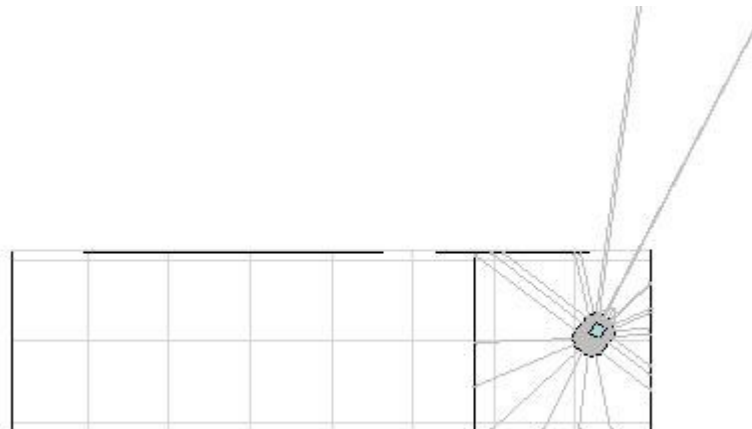
Sont précisés :

- la distance parcourue ou position odométrique (abscisse, ordonnées, orientation) ;
- les indications de vitesse (vitesse de translation et de rotation) ;
- la position absolue (abscisse, ordonnées, orientation).



Comme le montre la capture d'écran ci-dessous, un plan peut être chargé par le simulateur, qui permet de visualiser ainsi la gestion des collisions par le robot.

On peut d'ailleurs visualiser la diffusion des rangées de sonar arrière et avant, représenté ici par les traits grisés.



Un clic droit sur le robot permet de modifier son orientation, un clic gauche de modifier son emplacement. La roulette de la souris permet d'effectuer un zoom avant ou arrière sur le robot.

Il est à noter que le déplacement à la souris modifie les valeurs de la valeur de position absolue, mais en aucune façon les valeurs odométriques.

Il est néanmoins possible de réinitialiser la position du robot allant dans File/Reset ou en faisant Ctrl-R.



## V. APPLICATION CLIENTE

Comme je l'ai déjà dit plus haut, le comportement du robot est de type Client/Serveur. Il est donc possible de commander le robot à partir d'une application cliente.

Cette application peut être lancée à partir d'un poste client, ou directement à partir du système d'exploitation embarqué. Il est également envisageable d'avoir une structure mixte dans laquelle nous avons 2 applications clientes, l'une sur un poste distant et l'autre sur le robot, communicante entre elles et gérant des fonctionnalités différentes du robot.

### 1) **ARIA**

Pour réaliser cette application, on dispose d'un environnement de développement Open-Source nommé Aria (Activmedia Robotic Interface for Applications) qui permet de créer une interface efficace avec le robot et ses accessoires.

La librairie qu'elle contient est codée en langage C++. Il est cependant possible de programmer en d'autres langages tel que Java ou Python.

En plus de gérer les commandes du robot, la librairie Aria dispose également de différentes classes génériques sur la gestion des sockets, des threads, etc... Ces classes génériques reposent sur des fonctions répondant à la norme POSIX : les programmes développés à partir de ces classes pourront donc être indifféremment compilés sous Linux que sous Microsoft Windows.

La librairie ARIA est "open source", autrement dit elle est livrée avec l'ensemble des sources qui ont permis de la créer. Il est donc possible de modifier une de ses sources, ou même de rajouter une classe, puis de recompiler la librairie.

## 2) CYCLE DE FONCTIONNEMENT

### A) Principe

L'application cliente développée avec Aria exécute un cycle de fonctionnement, autrement dit un ensemble de processus qu'elle lance les uns à la suite des autres de façon synchrone.

En effet, comme le montre le schéma ci-contre, il est à noter que la récupération, l'interprétation et la restitution des mesures à l'application cliente font partie intégrante du cycle. Ainsi, si une tâche synchrone, une fonction ou une boucle, bloquent le cycle, alors l'application n'est plus renseignée des changements éventuels des mesures des capteurs. (Cf *TP Commande de la Pince*)

### B) Déclenchement

Par défaut, ce cycle est déclenché à chaque fois que l'application reçoit un SIP (Server Information Packet, c'est-à-dire l'ensemble des données provenant des capteurs) de la part du robot. L'application possède également son propre temps de cycle (100 ms par défaut) : si au bout de 2 temps de cycle, aucun paquet n'est arrivé du robot, alors elle relance le cycle automatiquement.

Ce procédé permet d'effectuer les actions désirés sur le robot ou un de ces périphériques en temps réel, autrement dit en fonction des informations les plus récentes envoyées par les capteurs.

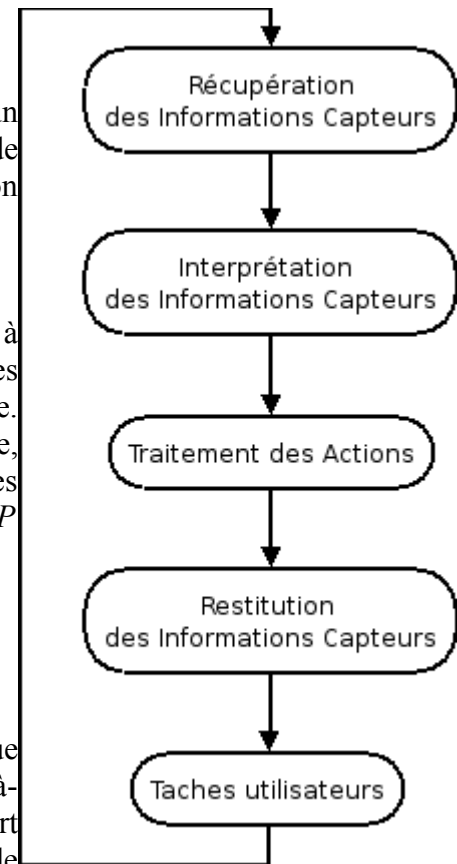
### C) Modes de Fonctionnement

Le cycle peut être lancé soit de façon synchrone (ArRobot::run()), ou en tant que nouvelle tâche de fond (ArRobot::runAsync()). Dans les deux cas, le cycle est arrêté par la méthode ArRobot::stopRunning()).

De plus, le cycle peut très bien fonctionner sans se connecter au robot (Cf *TP Pointeur de fonction*). Dans ce cas, le cycle n'est déclenché que par le temps de cycle propre à l'application.

On peut enfin forcer le robot à n'exécuter le cycle qu'une seule fois, en sachant que l'on ne pourra alors récupérer et utiliser les valeurs des capteurs.

Utiliser le cycle de fonctionnement est la façon la plus simple pour créer une application



### **3) DEPLACEMENTS**

On peut considérer qu'il y a 3 méthodes différentes pour gérer les déplacements :

- par décomposition des mouvements,
- suivi de lignes de couleur,
- par position.

#### **A) Décomposition des mouvements**

On peut décomposer les déplacements mouvement par mouvement.

On peut en effet faire avancer ou reculer le robot d'une distance précisée en millimètres. On peut également modifier l'orientation du robot exprimée en degrés.

Pour cela, la classe ArRobot dispose de plusieurs méthodes qui autorisent une gestion simple et basique des mouvements.

#### **B) Suivi de lignes de couleur**

Grâce à sa caméra embarquée, le robot est capable de suivre de façon autonome des lignes de couleur tracées au sol.

Pour plus de précision, se reporter au Chapitre IV.

#### **C) Positions**

Il est possible de faire déplacer le robot d'une position à une autre.

Les positions, définies par leur ordonnées, abscisses et orientations, peuvent être déterminées de façon absolue ou relative.

Pour la position absolue, la position initiale du robot est pris comme point de référence. La position courante du courant est alors calculée à partir de cette position à l'origine.

Quant à la position relative, la position courante est calculée par rapport à la position précédente.

### **4) ACTIONS**

#### **A) Principe**

Les actions permettent de pouvoir contrôler les déplacements du robot en fonction de son environnement. En effet, chaque action stipule le comportement que doit avoir le robot à un instant  $t$  selon les valeurs des capteurs reçues.

En pratique, une action est une classe dérivée de la classe ArAction. Une fois les spécifications de l'action implémentée dans la méthode virtuelle *fire()*, l'action est rajoutée au cycle de fonctionnement de l'application par la méthode *addAction()* de la classe ArRobot.

A l'intérieur d'un cycle, une action est spécifiée par un nom unique, sous la forme d'une

chaîne de caractères, et un ordre de priorité, de 0 à 100.

Bien entendu, on peut ajouter plusieurs actions dans un même cycle, chaque action décrivant une réponse à une situation différente (suivi d'un itinéraire, blocage des roues, évitement d'un obstacle...).

Cependant, pour être prise en compte dans le cycle, l'ensemble des actions doit être ajouté avant le démarrage du cycle. On peut alors désactiver une action inutile (méthode *deactivate()* de la classe *ArAction*) pour la réactiver par la suite (méthode *activate()*) lorsque la situation l'exigera.

## **B) Types d'actions**

On peut distinguer trois types d'actions, bien qu'il n'existe pas de classes pour les différencier :

- les actions de mouvements,
- les actions de limitations,
- les actions mixtes.

Les actions de mouvements stipulent les paramètres des mouvements à adopter pour atteindre un but précis : suivre une ligne, aller à un point précis...

A l'inverse, les actions de limitations assurent quant à elles la sécurité du robot en fixant des limites aux mouvements, évitant ainsi des collisions ou des accélérations et décélération trop brusques.

Enfin, les actions mixtes, généralement plus efficace, permettent de définir une trajectoire à adopter tout en s'assurant de la sécurité du robot.

Il est recommandé d'allouer une priorité plus élevée aux actions de limitations afin de s'assurer qu'un mouvement est sans risque avant de l'effectuer.

## **C) Action désirée**

Tout changement que l'on souhaite dans les déplacements du robot se traduit en pratique par la modification des paramètres contenus dans la classe *ArActionDesired*, qui sont ensuite transmis au robot.

Parmi ces paramètres, on trouve :

- la vitesse ;
- le cap relatif (delta heading), c'est-à-dire l'angle entre la position actuelle du robot et la position désirée ;
- le cap absolu (heading), c'est-à-dire l'angle entre la position initiale du robot et la position désirée ;
- la vitesse maximum en marche avant ;
- la vitesse maximum en marche arrière ;
- la vitesse maximum en rotation.

## **D) Asservissements**

Afin de contrôler que ces paramètres soient bien respectés, le robot dispose d'un PID interne pour assurer l'asservissement en translation et en rotation du robot.

Les paramètres du PID peuvent être modifiés par l'envoi de paquets de commandes.

### **E) Actions prédéfinies**

La librairie ARIA propose initialement une série d'actions prédéfinies que l'on peut utiliser en l'état et dont voici une liste non exhaustive :

ArActionAvoidFront/ArActionAvoidSide. : évitement d'un obstacle frontal ou latéral.

ArActionColorFollow : déplacement du robot vers une couleur située dans son champ de vision.

ArActionGoto : déplacement du robot vers une position pré-établie

ArActionRecover : série de tentatives effectuées lors du blocage d'une des roues.

ArActionStop : arrêt du robot

Bien que souvent rudimentaires, ces actions constituent un bon exemple de la création et l'utilisation des actions.

## **5) TACHES UTILISATEURS**

### **A) Principe**

Autant les actions servent à effectuer une tâche rudimentaire correspondant à une situation donnée, les tâches utilisateurs permettent d'effectuer les modifications nécessaires au comportement du robot, de façon à effectuer une tâche complexe. Par exemple : récupérer un objet et le rapporter.

La modification du comportement du robot se traduit simplement par l'activation et la désactivation des actions appropriées.

Il est possible d'ailleurs de définir plusieurs tâches utilisateurs, chacune d'elles pourra ainsi correspondre à un cas d'utilisation, principal ou secondaire du système.

Ainsi, c'est en général à l'intérieur des tâches utilisateurs que sera contenu le squelette de l'application.

### **B) Intégration dans le cycle**

A l'instar des actions, les tâches utilisateurs doivent être intégrées dans le cycle, par la méthode *addUserTask()* de la classe *ArRobot*. Elles possèdent également un nom unique ainsi qu'une priorité.

Le fait qu'une tâche utilisateur soit incorporée au cycle présente quelques contraintes : en effet, au lieu d'être exécutée de façon linéaire comme cela se passe traditionnellement pour toute application, la tâche est relancée à chaque redémarrage du cycle, soit environ toutes les 100 ms.

De ce fait, pour qu'une tâche puisse conserver une valeur d'un cycle sur l'autre, il faut impérativement que la variable soit globale, ou – de façon plus élégante – que la tâche et la variable soit une méthode et un attribut d'une même classe.

Pour cette même raison, il faut s'assurer que la tâche n'initialise qu'une seule et unique fois ses variables, sinon elles seront réinitialisées à chaque cycle.

Enfin, les tâches utilisateurs ne disposent pas comme les actions des méthodes *activate()* et *deactivate()* : il est donc impossible de les désactiver une fois placées dans le cycle. Dès lors, il faut veiller à « neutraliser » une tâche si celle-ci n'est plus ou pas encore utile.

### **C) Automate à Etats Finis**

#### **Définition.**

Une des solutions possibles pour décrire un cas d'utilisation dans une tâche utilisateur est de définir le système comme un automate à états finis.

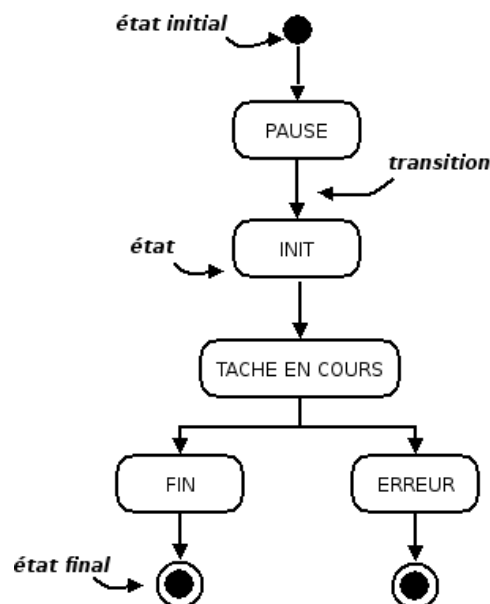
Un automate (ou machine) à états finis permet de décrire simplement le comportement d'un système en le décomposant en un nombre fini d'états distincts, reliés entre eux par des transitions. Les transitions peuvent éventuellement être soumises à une condition évaluée : position atteinte, réception d'un signal...

Chaque état peut correspondre alors à un ensemble d'activités afin d'atteindre un ou plusieurs objectifs (atteindre une position, démarré un système...), ou à un état précis du système (système en veille ou arrêter).

On notera qu'il est primordiale d'avoir un et un seul état initial, généralement définis dans le constructeurs de la classe. En revanche, il est tout à fait possible d'avoir plusieurs états finaux.

#### **Représentation.**

Le diagramme UML d'états/transitions est, avec le réseau de Pétri, un diagramme qui permet de représenter au mieux un automate à états finis.



#### **Etats particuliers.**

Certains états sont à remarqués plus particulièrement, du fait de leur utilité et de leur présence récurrente dans les automates à états finis :

- l'état **INIT** : état dans lequel se fait l'initialisation des variables nécessaires à la tâche,
- l'état **PAUSE** : état qui permet de neutraliser la tâche jusqu'à son activation,
- l'état **FIN** : état qui stipule que le but de la tâche est accompli,
- l'état **ERREUR** : état qui indique que la tâche n'a pu se dérouler normalement.

Les états INIT ou PAUSE pourront l'un ou l'autre constituer l'état initial de l'automate à états finis. De même, l'automate pourra avoir les états FIN et ERREURS comme états finaux.

### **Implémentation.**

Pour implémenter une tâche en utilisant le principe d'un automate à états finis, on a besoin de 2 éléments :

- une variable *statut* qui conservera l'état actuel de la tâche ;
- une structure conditionnelle de type *switch*, *qui*, à chaque lancement de la tâche, exécute le code associé à l'état actuel du système.

Exemple :

```
void MaTache()
{
    //Condition sur l'état
    switch(statut)
    {
        case INIT : //Initialisation des paramètres
                    break;

        case TACHE_EN_COURS : //code pour satisfaire la tâche à accomplir
                    break;
    }
}
```

L'ensemble des états possibles pourra être énuméré dans une variable de type *enum*, ou défini comme constantes grâce au mot clés *#define*. (La seconde possibilité est à préférée si cette liste d'états est vouée à s'agrandir.)

Exemple :    enum état {INIT, PAUSE,ERREUR}  
                   ou  
                   #define INIT 0  
                   #define PAUSE 1  
                   #define ERREUR 2

### **Structures Multitâche**

Lorsque l'on décompose l'application en plusieurs tâches utilisateurs, il existe 3 structures possibles pour les implémenter (la liste suivante est non-exhaustive) :

- structure parallèle,
- structure hiérarchique,
- structure mixte.

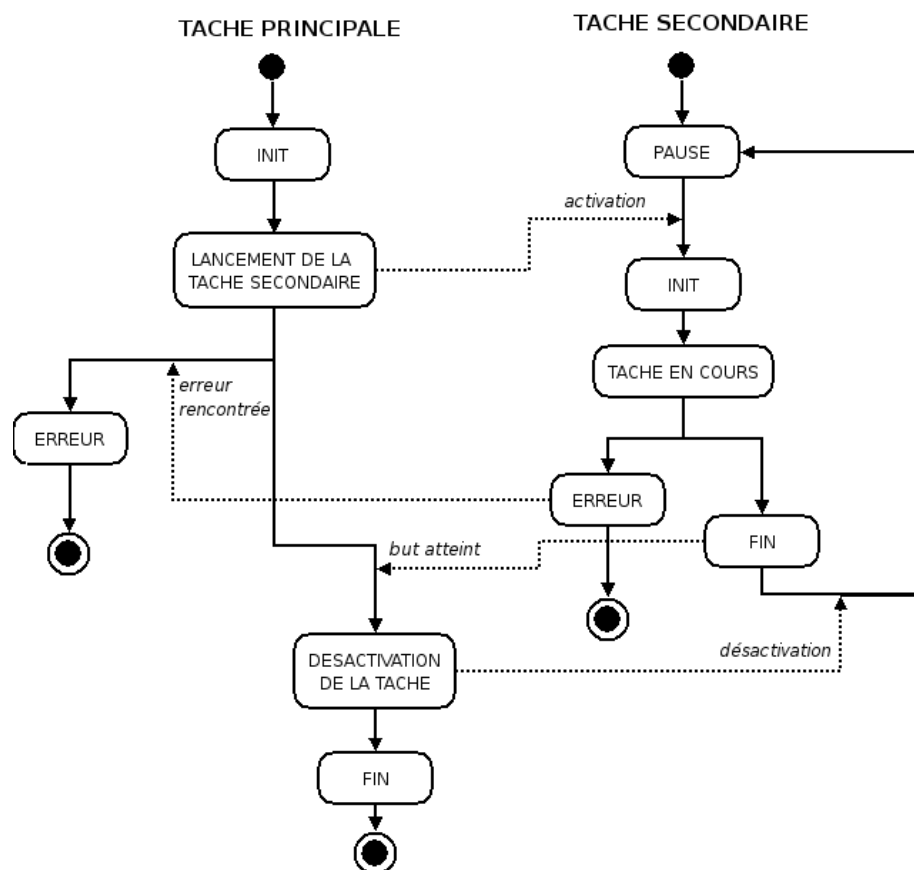
Dans une **structure parallèle**, les tâches sont toutes actives et utilisent la même variable *statut*. Ainsi, la modification de la variable par une des tâches a une incidence sur les autres tâches. Cette structure peut par exemple permettre de décomposer la tâche en plusieurs tâches au rôle bien précis, une tâche correspondant à un et un seul état. (dans ce cas, la structure conditionnelle *switch* peut être remplacée par une simple structure *if*.)

Un exemple est donné dans le TP *Commande de la pince d'un robot*. Dans ce TP, toutes les tâches sont en fait les méthodes d'un même objet et partagent le même attribut *statut*.

Dans une **structure hiérarchique**, seule la tâche principale est toujours active, et peut, selon la situation, activer ou désactiver des tâches secondaires. Dans ce cas, chaque tâche possède une variable statut propre, indépendante des autres. Cette structure peut permettre de décomposer le système en cas ou sous-cas d'utilisation.

La librairie Aria ne proposant des fonctions pour activer ou désactiver une tâche utilisateur, on utilisera un état PAUSE durant laquelle la tâche n'a aucune activité. (Pour les tâches secondaires, l'état PAUSE sera généralement l'état initial.)

En revanche, il faut prévoir des fonctions qui permettent à la tâche principale de gérer ses tâches secondaires : activer, désactiver, savoir si le but de la tâche est atteint...



Le diagramme ci-dessus est une représentation possible d'une application structurée hiérarchiquement, avec une tâche principale et une tâche secondaire. On peut ainsi voir que chaque tâche possède sa propre variable d'état, qui évolue donc de façon différente.

Les flèches en pointillés représentent les différentes interactions entre les 2 tâches et les termes en italique les conditions nécessaires pour que la transition s'opère.

Ces interactions permettent de montrer de quelle façon la tâche principale est en mesure de contrôler la tâche secondaire. Il est à noter que dans cet exemple, la tâche principale est en attente



de la fin de la tâche secondaire. Or, du fait de leur exécution en parallèle, il est également envisageable que la tâche principale puisse continuer de réaliser d'autres activités pendant que se déroule la tâche secondaire.

D'autre part, ce diagramme ne représente qu'une seule tâche secondaire, mais on pourrait concevoir des structures plus complexes avec plusieurs tâches secondaires : ces tâches pourraient être activées les unes après les autres ou simultanément, ou bien même appelées par une autre tâche secondaire.

Ce diagramme montre enfin un bon exemple de tâche que l'on peut appeler plusieurs fois lors d'une même exécution. En effet, on peut constater que l'évolution de l'état de la tâche secondaire est cyclique et permet ainsi d'être à nouveau activée après s'être terminée une première fois. Il est alors possible de créer des fonctions qui modifient les paramètres de la tâche entre deux activations : modification de la position, du niveau à atteindre...

En dernier lieu, on peut avoir une **structure mixte** avec une tâche principale et des tâches secondaires, qui seront ensuite elles-mêmes décomposées en tâches parallèles.

## 6) COMMUNICATION TCP/IP

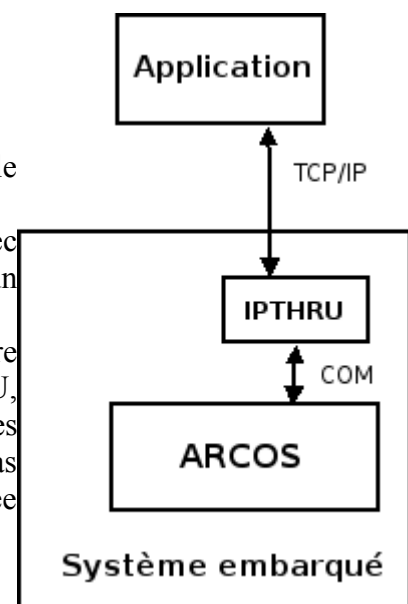
### A) Pont TCP/Série

Les communications avec ARCOS se font par défaut via le port série, qu'il soit externe ou bien interne.

Ainsi, si une application cliente veut communiquer avec ARCOS via le protocole TCP/IP, elle est obligée de passer par un pont logiciel entre TCP/IP et le port série.

La librairie ARIA fournit à ces fins, dans le répertoire */usr/local/Aria/examples*, le code de l'application IPTHRU, application qui se comporte en serveur TCP et retransmet les informations reçus sur le port série interne. Cette application n'est pas lancée par défaut au démarrage du robot, et doit donc être lancée manuellement ou rajoutée à *inetd*.

Le port par défaut est 8101.



### B) Choix du protocole de Transport

La librairie ARIA propose les 2 protocoles UDP et TCP comme protocoles de la couche Transport, le protocole TCP étant par défaut.

Cependant, le protocole TCP n'est pas forcément le plus adapté pour une application temps réel. En effet, les QoS proposées par TCP ont pour conséquence un ralentissement dans la transmission.

Par exemple, si un paquet est perdu (ce qui n'est pas forcément important dans l'application cliente), le protocole TCP va alors redemander ce paquet. Or le temps de le redemander et que le paquet soit ré-acheminé, les valeurs qu'il contient sont peut-être devenues obsolètes.

En revanche, le protocole UDP qui ne propose aucune QoS, est de fait plus rapide. Néanmoins, ce protocole n'assure pas l'ordonnancement des paquets transmis. Ce problème peut se produire lorsque les paquets sont acheminés via Internet et transitent par des serveurs différents. Dans le cas d'un réseau local, le problème ne se pose.

Si on souhaite toutefois utiliser le protocole UDP au travers d'Internet, il est possible le protocole RTP (Real-time Transfert Protocole) situé sur la couche Session. Ce protocole, utilisé pour la transmission temps réel de sons et de vidéos, permet le séquençement des paquets et d'y ajouter un marqueur temporel.

## 7) **COMPILATION**

### **A) Compilation sous LINUX**

Sous Linux, il est possible de compiler un projet grâce au compilateur g++ ou gcc. Il faut veiller à la compilation de bien préciser l'emplacement de la bibliothèque Aria, notamment lors de l'édition de liens.

Voici un exemple de fichier Makefile, permettant de compiler un projet basé sur la librairie Aria :

```
%: %.cpp Robot.o CommandePince.o
    g++ -g -Wall -D_REENTRANT -fno-exceptions -Iinclude Robot.o
    CommandePince.o -o $@ $< -L/usr/local/Aria/lib -lAria -lpthread -ldl

Robot.o: Robot.cpp Robot.h
    g++ -g -Wall -D_REENTRANT -fno-exceptions -Iinclude -c Robot.cpp

CommandePince.o : CommandePince.cpp CommandePince.h
    g++ -g -Wall -D_REENTRANT -fno-exceptions -Iinclude -c CommandePince.cpp
```

Le projet de cet exemple comprend un fichier principal - où se trouve la fonction main() - fichier qui fait lui-même référence deux « include » : Robot.cpp et CommandePince.cpp

Dans ce fichier MakeFile, le nom du fichier principal n'est pas précisé (remplacé par le symbole %), ce qui permet de le choisir librement. Pour compiler, on doit alors exécuter l'instruction *make nom\_pgme*, où nom\_pgme est le nom du fichier sans l'extension.

Ex: > make Client

Les instructions de compilation, basées sur le compilateur g++, s'exécutent après vérification que les fichiers aient subis des modifications depuis la dernière compilation.

Le paramètre *-Iinclude* permet d'inclure les fichiers contenu dans le dossier include, alors que les paramètres *-L/usr/local/Aria/lib -lAria* permettent d'établir le lien avec la librairie Aria.lib situé dans le répertoire /usr/local/Aria/lib.

## **B) Compilation sous MICROSOFT WINDOWS**

### **Microsoft Visual C++ .NET 2003**

Pour pouvoir compiler un projet se servant de la librairie Aria, il est indispensable que le projet soit correctement configuré :

Créer tout d'abord un nouveau projet (*File->New->Project...*) en choisissant "Console Application".

Aller modifier les propriétés en allant dans *Project->Properties*.

Dans le champ "Configuration" (en haut à gauche), choisir "All Configurations".

Cliquer sur "Linker"

Dans la section "General", au niveau de "Additional Library Directories", ajouter le chemin de la librairie Aria (C:\Program Files\ARIA 2.4-0\lib).

Dans la section "Input", dans "Additional Dependencies", ajouter les librairies suivantes : "Aria.lib winmm.lib wsock32.lib"

Cliquer sur C/C++

Dans la section "General", au niveau de "Additional Include Directories", ajouter le chemin du fichier Aria.h (C:\Program Files\ARIA 2.4-0\include).

Dans la section "Precompiled Headers", au niveau de "Create/Use Precompiled Header", choisir "Not Using Precompiled Headers".

Aller à la section "Code Generation".

Changer le champ "Configuration" sur "Debug", puis à la rubrique "RunTime Library", choisir "Multi-threaded Debug".

Changer à nouveau le champ "Configuration" sur "Release", puis à la rubrique "RunTime Library", choisir "Multi-threaded".

### **Microsoft Visual C++ EXPRESS 2005**

Le compilateur Microsoft Visual C++ EXPRESS 2005 a comme avantage par rapport à la version précédente d'être en téléchargement gratuit pendant un an. En contrepartie, il ne gère ni les MFC et ni les ATL, et ne dispose pas de la plate-forme SDK par défaut. (Celle-ci peut cependant être installée séparément.)

Ainsi, la configuration du compilateur est la même que la précédente mais avec la nécessité de rajouter les chemins pour la librairie SDK.

Dans la section "C/C++ → General", au niveau de "Additional Include Directories", ajouter le chemin C:\Program Files\Microsoft SDK\include.

Dans la section "Linker → General", au niveau de "Additional Library Directories", ajouter le chemin C:\Program Files\Microsoft SDK\Lib.

Si vous compilez, vous pouvez apercevoir des warnings provenant de la librairie Aria. On peut en éliminer un bon nombre en ajoutant en début du programme la définition suivante :

```
#define _CRT_SECURE_NO_DEPRECATED
```

### **C) Compilation de la bibliothèque dynamique.**

Etant donné que la librairie Aria.dll est open source, il est possible de recompiler entièrement la librairie.

Ceci peut se révéler utile en cas de modifications d'une partie du code ou de l'adaptation de la librairie au système d'exploitation.

## **8) JAVA**

Comme je l'ai déjà précisé, il est possible d'utiliser ARIA sous différents langages de programmation - et notamment sous JAVA.

### **A) JNI**

Pour pouvoir utiliser la librairie ARIA codée en C++ à l'intérieur d'une application JAVA, les développeurs ont utilisé la technologie JNI (JAVA Native Interface).

Grâce à JNI, il est possible d'appeler des méthodes dites "natives" codée en C++ à partir de prototype JAVA. Ceci permet de développer en JAVA, et de coder certaines méthodes en C++ quand ce langage se trouve être le plus adapté (programmation bas niveau, temps réel...). En contre partie, le code JAVA généré dépend maintenant du code C++ et perd en portabilité. (On risque de trouver ainsi des dysfonctionnements si on désire développer une applet reposant sur la librairie ARIA.)

Il est cependant à noter que la librairie sous JAVA n'est pas autant testé que celle sous C++, et donc que des erreurs peuvent être générées par la librairie.

### **B) Limitation**

Il est à noter que l'utilisation de JNI entraîne une perte de certaines caractéristiques de la librairie ARIA. En outre, il n'est pas possible de surcharger en JAVA les méthodes virtuelles.

Ceci a pour conséquence l'impossibilité de créer de nouvelles actions à partir de la classe ArAction. Si cette création est cependant nécessaire, il faut créer cette nouvelle classe en C++, l'incorporer à la librairie et générer enfin son interface JAVA.

### **C) Compilation**

Pour la compilation en Java, on peut tout simplement utiliser l'environnement Java 2 SDK (Standard Edition version 1.4.2), disponible sous Linux ou Windows et développé par Sun Microsystems (TM).

Ce kit de développement propose un compilateur un mode lignes de commande :

- le programme *java* permet de transformer les classes en fichiers objets ;
- le programme *javac* quant à lui, réalise un exécutable à partir des fichiers objets.

## 9) CREATION D'UNE IHM

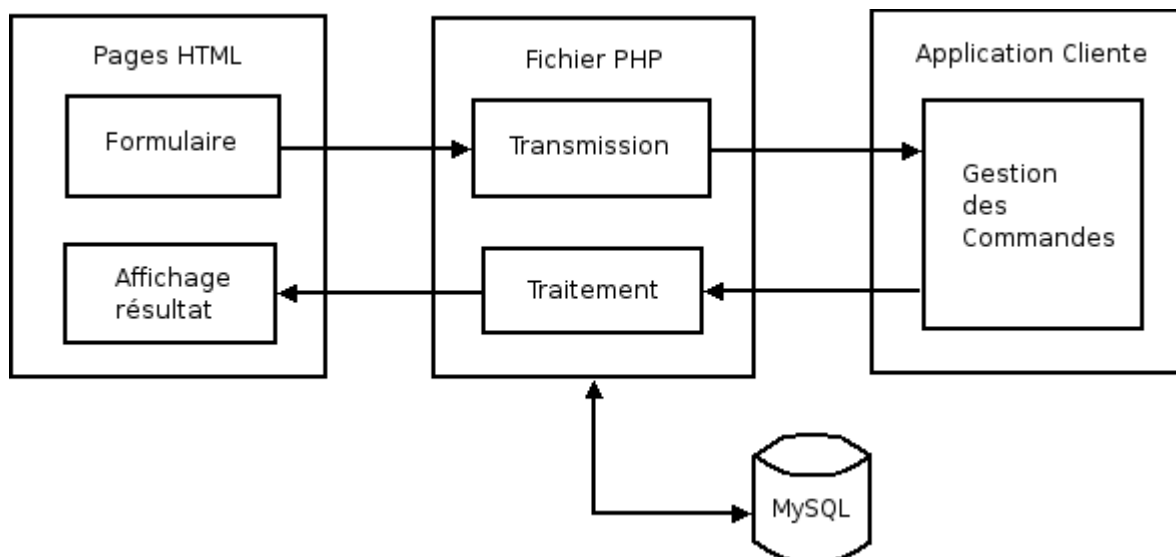
Bien que ce soit un langage rapide, adapté aux applications matérielles, le langage C++ ne permet de créer facilement une Interface Homme-Machine ergonomique.

Cependant, il est possible de réaliser des ponts entre l'application cliente destinée à commander le robot et une IHM écrite dans un autre langage, plus adapté à la création d'interfaces.

### A) Les pages Web dynamiques

Le langage de présentation HTML (HyperText MetaLanguage) a été initialement créé pour réaliser facilement des pages, consultables sur Internet et pouvant recevoir des contenus variés : textes, images, sons, vidéos... Ces pages se trouvent sur un serveur HTTP et sont par conséquent accessibles par tout ordinateur situé sur le même réseau.

Le langage interprété PHP (Preprocessor) permet, quant à lui, de récupérer et de traiter les informations envoyées depuis un formulaire HTML, de les transmettre à l'application cliente, puis de modifier les pages HTML en fonction des informations retournées par l'application.



De plus les programme PHP dispose de modules efficaces pour dialoguer avec une base de données (MySQL par exemple) afin de stocker certains paramètres ainsi que les échanges entre l'IHM et l'application cliente.

### B) Un interface JAVA

Il ne s'agit plus d'écrire tout le programme en JAVA comme au paragraphe V.8, mais uniquement l'IHM, le cœur de l'application cliente demeurant en C++. Nous créons ainsi une application répartie, en utilisons dans chaque domaine le langage de programmation le plus adapté : C++ pour la partie matérielle, JAVA pour l'IHM.

Le problème réside à présent dans le dialogue bi-directionnel entre deux applicatifs, écrits dans deux langages différents. La solution que je propose est d'utiliser un *middleware* (de *middle* = milieu et *software* = logiciel), un logiciel intermédiaire entre deux processus.

## CORBA

Créé par l'OMG (également inventeur de l'UML), CORBA (Common Object Request Broker Architecture) est un middleware qui permet de créer des objets distribués entre plusieurs applications, écrits en langages différents et/ou s'exécutant sous des environnements différents.

CORBA repose sur une architecture Client/Serveur : le serveur instancie un ou plusieurs objets, puis les met à la disposition d'applications clientes au travers d'un ORB (Object Request Broker). Les clients peuvent donc se connecter à l'ORB et faire ensuite appel aux méthodes de l'objets.

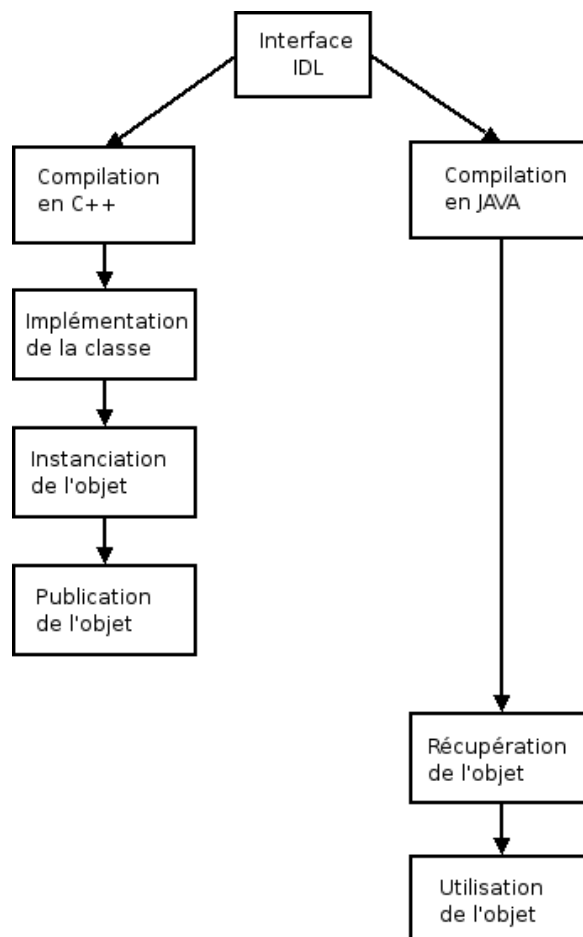
Cette technologie a été normalisée, et par conséquent, toutes les bibliothèques (gratuites et payantes) qui l'implémentent sont compatibles entre elles. Pour preuve, on peut utiliser dans notre cas le compilateur IDL fourni avec le Java SDK 2, et pour la partie C++ l'ORB gratuit MICO.

## IDL

Le langage IDL (Interface Definition Language) est, comme son nom l'indique, un langage permettant de définir l'interface qui sera utilisé par la partie cliente et implémentée par la partie serveur;

En effet, pour que le client connaisse les méthodes susceptibles d'être appeler pour un objet considéré, ces méthodes doivent être au préalable définies dans une interface. Les méthodes appelées étant exécutées côté serveur, le serveur se doit d'implémenter la classe selon l'interface définie.

Le schéma ci-dessous montre les différentes étapes à réaliser pour créer une application répartie entre un serveur en C++ et un client en JAVA, tous deux utilisant un même objet :



## VI. AUDIO/VIDEO

Afin de mieux communiquer avec son environnement, le robot PeopleBot dispose d'une caméra PTZ couleur, d'un microphone et de haut-parleurs ainsi qu'une série d'utilitaires permettant de les exploiter.



### 1) **CONVERSION VIDEO**

Le système dispose d'un « framegrabber » embarqué (du verbe *to grab* : saisir et de *frame* : cadre), c'est-à-dire d'un convertisseur des informations des capteurs CMD en images vidéos.

Les logiciels de traitement et de transmissions vidéos doivent par conséquent être lancés en tant que serveur sur le système, et ce afin de récupérer l'image convertie par le framegrabber.

### 2) **MANIPULATION DE LA CAMERA**

Reliée par liaison série, la caméra embarquée sur le robot est de type PTZ (Pan-Tilt-Zoom), c'est-à-dire que l'on peut effectuer dessus trois types de réglage :

- orientation panoramique ;
- inclinaison;
- zoom.

La classe ArVCC4 permet de manipuler la caméra à partir d'une application cliente, et ce sur les 3 axes de liberté possibles.

Il est à noter que la manipulation de la caméra dépend du micro-contrôleur ou du PC auquel il est relié, et en aucun cas du framegrabber (Cf. chapitre précédent). La gestion des mouvements de la caméra et la récupération des images sont ainsi deux actions complètement indépendantes.

Cette manipulation tridimensionnelle de la caméra permet d'obtenir une vue panoramique de l'environnement lors d'une phase statique du robot. Cette vue peut ensuite servir à l'orientation du robot (par suivi de ligne, par exemple) ou à la reconnaissance d'objet présent dans son champ de vision.

### 3) **RECONNAISSANCE D'IMAGE**

Avec le robot, est fourni l'utilitaire ACTS (ActivMedia Color Tracking System) qui permet de détecter une ou plusieurs couleurs prédéfinies dans l'image transmise par la caméra.

Ceci permet de reconnaître d'un objet au sein d'un environnement ou de gérer les déplacements du robot.

### **A) Phase d'entraînement**

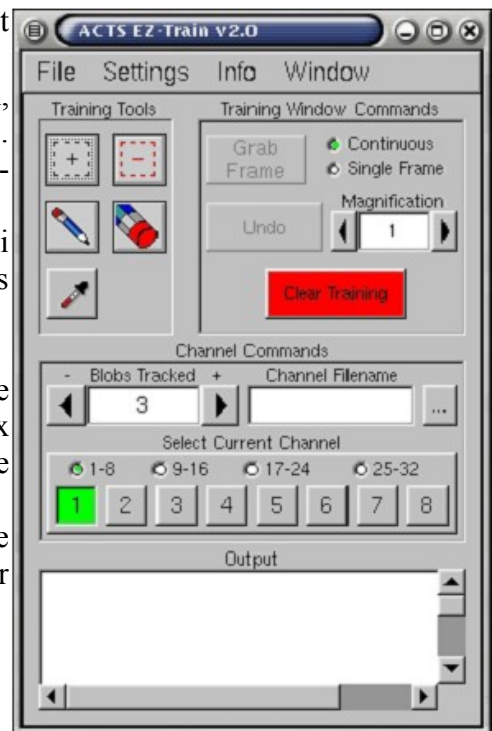
Avant d'exploiter ACTS, une phase d'entraînement est nécessaire afin de déterminer les couleurs à rechercher.

Cette phase est réalisée grâce à l'utilitaire EZ-Train, application cliente lancé en même temps que le serveur ACTS. (Cet utilitaire ne peut être ouverte que sous environnement X-Window.)

Pour cela, il propose une console de commandes ainsi qu'une fenêtre d'acquisition dans laquelle se placent les images issues de la caméra.

Avant de sélectionner une couleur, il est nécessaire de choisir un canal parmi les 32 possibles. Ces canaux indépendants correspondent chacun à la recherche d'une couleur précise.

Le canal choisi, on peut le nommer, mais aussi fixer le nombre de « tâches » - de 1 à 10 - à rechercher avec la couleur choisie.



Une fois ces réglages établis, on pointe la caméra vers la couleur à détecter et on la sélectionne avec la souris.



Si le résultat paraît satisfait, il faut alors enregistrer le canal et passer à un autre.

Une fois que l'ensemble des canaux voulus sont enregistrés et la caméra correctement paramétrée, on peut alors enregistrer la configuration.

Attention : enregistrer une configuration **n'enregistre pas** les canaux associés !

### **B) Serveur ACTS**

Une fois la phase d'entraînement effectuée et les canaux enregistrés, on peut alors lancer une version allégée du serveur ACTS – sans EZ-Train – qui tournera en arrière plan.

Ce serveur s'ouvre grâce à la ligne de commande suivante, où *ma\_config* est une



configuration enregistrée dans la phase d'entraînement :

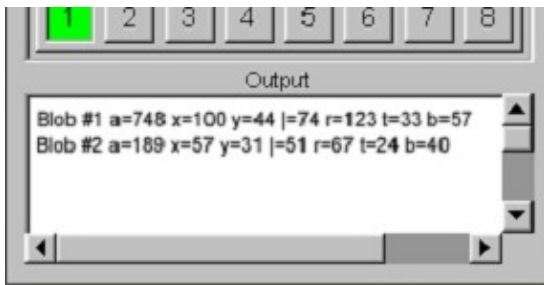
`% ./acts -t ../ma_config`

Cette version du serveur ne nécessite pas l'environnement X-Window et est par conséquent moins gourmande en ressources système.

Le port par défaut est le 5001.

A partir des informations contenues dans la configuration, ACTS va identifier, dans les images diffusées par la caméra, des « tâches de couleur » ou « blob », qui sont en fait des amas de pixels de la couleur désirée. (Les tâches inférieures à 10 pixels sont considérées comme du bruit et donc ignorées.)

Pour chaque tâches détectées, ACTS transmet alors des paramètres de taille et de position aux applications clientes connectées :



- centre de gravité (x,y)
- aire (nombre de pixels)
- bordures extérieures (haut, bas, gauche, droite)

Les bordures extérieures correspondent aux côtés d'un rectangle qui entoure la tâche.

### **C) Intégration dans une application cliente**

La librairie ARIA dispose de classes écrites en C++ qui permettent de récupérer les informations émises par le serveur ACTS.

La classe `ArACTS_1_2` permet de se connecter via TCP/IP et récupérer ensuite les informations des blobs d'un canal donné.

La classe `ArACTSBlob` permet quant à elle de gérer les informations d'un blob.

Un exemple de cette utilisation est donné dans la fonction `ArActionColorFollow`. Cette fonction fait déplacer le robot en direction d'une couleur détectée par la caméra.

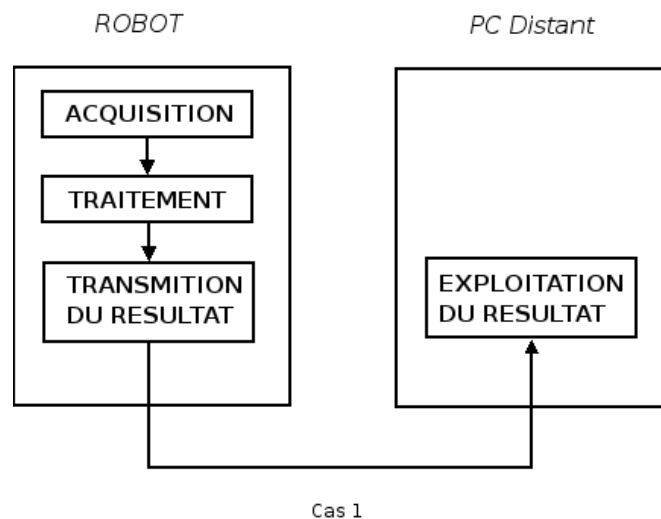
## **4) TRAITEMENT DE L'IMAGE**

### **A) Exploitation des ressources**

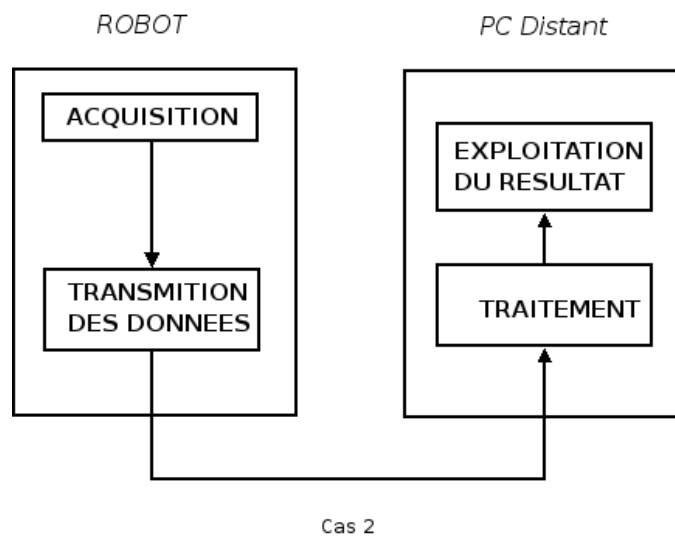
Le traitement de l'image, et a fortiori d'une vidéo, peut se révéler « gourmand » en ressources. En effet, compte tenu de la taille des données à traiter et de la vitesse de rafraîchissement de l'image, le traitement d'une vidéo peut accaparer une grande partie de la mémoire vive ainsi que de la charge du processeur.

Il peut donc être important de se demander si on souhaite exécuter un tel traitement à l'intérieur même de l'ordinateur embarqué du robot ou de l'extérioriser sur un PC distant, susceptible d'être plus à même de traiter l'information.

Deux solutions s'offrent à nous :



Le cas 1 montre un traitement effectué dans l'ordinateur embarqué, ce qui risque d'amoinrir les performances générales du système. Le résultat (position d'un objet, détection d'un mouvement...) peut alors être transmis



Pour le cas 2, le traitement s'effectue, non plus dans le robot, mais sur un PC distant, plus adapté à ce genre de travail.

Néanmoins, ce cas soulève un autre problème, car, comme la vidéo transite du robot vers le PC, cela risque d'augmenter la charge du réseau et d'affecter la bande passante. Ce problème peut être réglé en surdimensionnant le débit du réseau.

## **B) Librairies**

Deux librairies en C++ sont disponibles pour la récupération et le traitement de l'image : VisLib de la société ActivMedia Robotics/MobileRobots, Inc. et OpenCV de la société Intel.

La librairie Vislib ne fonctionne que sous Linux, et se révèle adéquate pour une application embarquée.

La librairie OpenCV, plus complexe, fonctionne uniquement sous les architectures Intel et le système d'exploitation Windows.

## **5) SYNTHESE VOCALE**

Les buzzers disponibles sur le robot PeopleBot peuvent servir à un module de synthèse vocale, c'est-à-dire l'imitation de la voie humaine par une machine.

Ceci pourra permettre d'améliorer l'IHM avec un support non plus uniquement visuel (écran d'un ordinateur) mais également auditif.

Un application concrète peut être lors de la rencontre avec un obstacle, que le robot puisse « exprimer » une requête (polie, bien entendu) afin que, si l'obstacle est humain, il s'écarte pour laisser le robot passer.

## **6) RECONNAISSANCE VOCALE**

Le robot PeopleBot dispose de deux microphones, situés de part et d'autre du sommet. Il peuvent être utiliser afin d'écouter les bruits issus de l'environnement du robot, mais également dans le cadre de reconnaissance vocale, c'est-à-dire la traduction en commande d'un ordre formulé vocalement.

Cet aspect complète parfaitement l'IHM auditif commencée avec la synthèse vocale.

Une des applications les plus utiles pourrait la reconnaitre par le robot de la phrase « stop » en tant que formule d'arrêt d'urgence.

## **7) TRANSMISSION DU SON ET DE L'IMAGE**

Les images de la camera sont transmises à un « framegrabber », applicatif logiciel qui permet la conversion des données issues de la caméra en image vidéo exploitable.

Le logiciel SAV (Server Audio Video) est destiné à restituer les images issues de la caméra, mais aussi les sons recueillis par les deux microphones dont dispose le robot. Une fois le serveur SAV lancé, il possible de visualiser les images grâce au client SAV ou tout autre logiciel incluant la vidéo tel que MobilEye.

Cependant, pour transmettre la vidéo à un client distant, il est nécessaire de passer par l'intermédiaire d'une application serveur dédiée basée sur ArNetworking. Ce serveur, grâce à la classe ArHybridForwarderVideo, récupère les images sur le serveur d'image puis les met à la disposition à tous client qui souhaite s'y connecter.

La récupération des images peut être, par exemple, au centre d'un système autonome de surveillance d'un entrepôt ou plus généralement d'une entreprise en dehors de sa période d'activité.

## VII. RESOLUTION DE PROBLEMES

Ce chapitre présente plusieurs erreurs techniques susceptibles d'être rencontrées et explique pour chacune d'entre elles la façon de la résoudre.

### 1) Erreur au démarrage de MobileSim sous Windows XP

Lors du démarrage de la simulation, avec ou non chargement d'un fichier .map, l'erreur suivante risque d'arriver :

"C:\DOCUME~1\Temp\MobileSim-stage\_world-3168.world:51 : syntax error"

Ce problème vient d'une incompatibilité entre le logiciel et le système d'exploitation : en effet le logiciel attend le symbole "." comme séparateur des décimales au lieu de ",". Pour résoudre le problème, il faut faire la démarche suivante :

- Aller dans *Démarrer => Panneau de configuration => Options Régionales et Linguistiques* ;
- Cliquer sur *Personnaliser* ;
- Modifier dans *Symbole Décimal* le symbole "," par "." ;
- Cliquer sur *OK*.

>> Ce bug est corrigé dans la version 0.2 de MobileSim.

### 2) Erreur à l'exécution d'un programme utilisant la librairie Aria sous Linux.

Lors de l'exécution d'une application embarquée sur le robot (ou sur un poste distant sous linux), il se peut que l'erreur suivante survienne :

```
peio@WILL-COYOTE:/usr/local/Aria/examples$ ./ipthru
./ipthru: error while loading shared libraries: libAria.so: cannot open share
d object file: No such file or directory
```

Le système d'exploitation ne trouve pas les librairies nécessaires à l'exécution, que ce soit libAria.so comme dans l'exemple ou libstdc++6.so.

Pour remédier à ce problème, il faut compléter le fichier *ld.so.config* avec le chemin absolu de la librairie manquante (ici */usr/local/Aria/lib*), puis exécuter en mode super-utilisateur la commande */sbin/ldconfig*.

NB : sur le système d'exploitation du robot, le fichier *ld.so.config* se trouve dans le répertoire */usr/local/gcc-3.4.1/lib*.

Pour d'autres OS de type Linux, on peut retrouver le fichier grâce à la commande:

*locate ld.so.config*

### 3) Erreur de connexion d'une application à MobileSim

Lorsque l'on souhaite qu'une application (par exemple demo.exe) se connecte à MobileSim,

il peut survenir le message suivant :

```
>> demo -rh localhost -ris
Connect to remote host localhost through tcp

syncing 0
syncing 1
syncing 2
Connect to robot.
Name: MobileSim
Type: Pionner
Subtype: peoplebot
Error: Have no parameters for this robot, bad
configuration or out of date Aria
Failed to connect to robot
Could not connect to robot... exiting
```

Cela vient d'une mauvaise gestion du modèle PeopleBot par le logiciel MobileSim. Pour y remédier, il faut modifier le fichier *PioneerRobotModels.world.inc*, contenu dans le répertoire */usr/local/MobileSim* sous Linux ou *Program Files\ActivMedia Robotics\MobileSim* sous Windows : à la place de "peoplebot", marquer "peoplebot-sh".

Puis, au lancement de MobileSim, il faut choisir le modèle "peoplebot-sh".

On pourrait également choisir directement le modèle "p3dx", qui se rapproche énormément de celui du PeopleBot.

>> Ce bug est corrigé dans la version 0.2 de MobileSim.

#### **4) Erreur à la compilation avec Aria sous Linux**

Lors de la compilation sous Linux, il se peut qu'une erreur soit générée à cause de l'absence de la librairie libstdc++.so.6, nécessaire à la librairie libAria.so.

Pour pallier ce problème, il est possible de créer un lien symbolique qui pointerait sur une ancienne version de la librairie libstdc++, grâce à la fonction *ln*.

Ce lien permet de forcer la librairie libaria.so à utiliser la version précédente de la librairie C++ au lieu de la version 6.

ex : `ln -s /usr/lib/libstdc++.so.5 /usr/lib/libstdc++.so.6`

D'autres erreurs peuvent également survenir provenant de la librairie ARIA :

```

/usr/local/Aria/lib/libAria.so: undefined reference to
`std::_Rb_tree_decrement(std::_Rb_tree_node_base*)'
/usr/local/Aria/lib/libAria.so: undefined reference to `std::_List_node_base::unhook()'
/usr/local/Aria/lib/libAria.so: undefined reference to
`std::_List_node_base::hook(std::_List_node_base*)'
/usr/local/Aria/lib/libAria.so: undefined reference to
`std::_Rb_tree_decrement(std::_Rb_tree_node_base const*)'
/usr/local/Aria/lib/libAria.so: undefined reference to
`std::_Rb_tree_insert_and_rebalance(bool,std::_Rb_tree_node_base*,std::_Rb_tree_node_base*,
std::_Rb_tree_node_base&)'
/usr/local/Aria/lib/libAria.so: undefined reference to
`std::_Rb_tree_rebalance_for_erase(std::_Rb_tree_node_base*,std::_Rb_tree_node_base&)'
/usr/local/Aria/lib/libAria.so: undefined reference to
`std::_Rb_tree_increment(std::_Rb_tree_node_base*)'
collect2: ld returned 1 exit status
make: *** [Client] Erreur 1

```

Ces erreurs proviennent directement d'une incompatibilité avec le compilateur. En effet, la librairie libaria.so a été initialement compilée avec la version 3.4 de g++. A partir de là, son utilisation avec une version précédente du compilateur générera obligatoirement des erreurs.

Pour remédier à ce problème, deux solutions est envisageable :

- mettre à jour le compilateur par un version plus récente ;
- recompiler la librairie avec la version du compilateur présente dans le système.

## **5) Erreur de liens à la compilation de la librairie sous Visual C++**

Lors de la compilation de la librairie dynamique Aria.dll avec le compilateur Visual C++ Express 2005, les erreurs de lien suivantes peut éventuellement survenir :

```

ariaUtil.obj: error LNK2019: unresolved external

symbol __imp__RegEnumValueA@32 referenced in function
"public: static bool __cdecl ArUtil::getStringFromRegistry(enum ArUtil::REGKEY,char
const *,char const *,char *,int)"
(?getStringFromRegistry@ArUtil@@SA_NW4REGKEY@1@PBD1PADH@Z)

ariaUtil.obj: error LNK2019: unresolved external
symbol __imp__RegQueryInfoKeyA@48 referenced in function "public: static bool __cdecl
ArUtil::getStringFromRegistry(enum ArUtil::REGKEY,char const *,char const *,char *,int)"
(?getStringFromRegistry@ArUtil@@SA_NW4REGKEY@1@PBD1PADH@Z)

ariaUtil.obj: error LNK2019: unresolved external
symbol __imp__RegOpenKeyExA@20 referenced in function "public: static bool __cdecl
ArUtil::getStringFromRegistry(enum ArUtil::REGKEY,char const *,char const *,char *,int)"
(?getStringFromRegistry@ArUtil@@SA_NW4REGKEY@1@PBD1PADH@Z)

../bin/ARIADebug.DLL: fatal error LNK1120: 3 unresolved externals

```

erreurs proviennent de l'absence d'une librairie à l'intérieur du projet. Pour régler ce problème, il faut donc rajouter la librairie ADVAPI32.lib dans les propriétés du projet (Linker/Input/Additionnal Dependencies).

## ***6) Problème de DLL à l'exécution d'un programme.***

Au lancement d'un programme sous windows, l'application peut s'arrêter brutalement par l'erreur suivante :

« Cette application n'a pas pu démarrer car ARIADebug.DLL est introuvable. La réinstallation de cette application peut corriger ce problème. »

On peut résoudre ce problème en plaçant la librairie concernée dans le répertoire « C:\Windows\system ».



# LEXIQUE

ACTS : ActivMedia Color Tracking System  
AFRI : Association Française de Robotique Industrielle  
ARCOS : ActivMedia Robotic Control and Operation Software  
ARIA : ActivMedia Robotic Interface for Applications  
CMD : Charge Modulated Device  
CORBA : Common Object Request Broker Architecture  
DHCP : Dynamic Host Configuration Protocol  
DLL : Dynamic Link Library  
GNU : GNU's Not UNIX  
GPL : *General Public licence*, Licence Publique Générale  
GUI : Graphic User Interface, interface graphique  
HTML : HyperText MetaLanguage  
HTTP : HyperText Transfert Protocol  
IDL : Interface Definition Language  
IHM : Interface Homme-Machine  
IP : Internet Protocol  
IPC : InterProcessus Communication  
PIXEL : *PICTure ELe ment*, élément d'image  
SDK : Software Development Kit  
JIRA : Association Japonaise des Robots Industriels  
JNI : JAVA Native Interface  
OMG : Object Management Group  
ORB : Object Request Broker  
OS : *Operating System*, système d'exploitation.  
PHP : Hypertext Preprocessor  
PID : Proportionnel Intégrateur Dérivateur.  
PTZ : Pan/Tilt Zoom  
QoS : *Quality of Services*, Qualité de Service  
RTP : Real-time Transfert Protocol  
SDK : Software Development Kit  
SIP : Serveur Information Packet  
SQL : Structured Query Language  
TP : Travaux Pratiques  
UML : Unified Modeling Language  
VNC : *Virtual Network Computing*, logiciel de contrôle à distance  
WEP : Wired Equivalent Privacy  
WiFi : Wireless Fidelity

## REFERENCES LOGICIELLES

- ACTS
- ARCOS
- ARIA 2.4.0
- Java 2 SDK
- MapperBasic 3
- Mico
- MobileEyes 1.2.8
- MobileSim 0.2
- OpenCV
- SAV
- UltraVNC
- VisLib
- Visual C++ Express 2005
- VNC

## DOCUMENTATION

- PeopleBot Manual V6
- ARIA Reference
- ACTS User Manual
- TP Commande multi-tâche de la pince d'un robot
- TP Commande des déplacements d'un robot

## ANNEXE 1 : Tableau des codes de commande

COMMAND	#	ARG S	DESCRIPTION	ARCOS VERSION
			<i>Before Client Connection</i>	
SYNC0	0	none	Start connection. Send in sequence. ARCOS echoes synchronization commands back to client, and robot-specific auto-synchronization after SYNC2.	1.0
SYNC1	1	none		
SYNC2	2	none		
			<i>After Established Connection</i>	
PULSE	0	none	Reset server watchdog.	1.0
OPEN	1	none	Start up servers.	1.0
CLOSE	2	none	Close servers and client connection.	1.0
POLLING	3	str	Change sonar polling sequence.	1.0
ENABLE	4	int	1=enable; 0=disable the motors.	1.0
SETA	5	int	Set translation acceleration, if positive, or deceleration, if negative; in mm/sec <sup>2</sup> .	1.0
SETV	6	int	Set maximum/move translation velocity; mm/sec.	1.0
SETO	7	none	Reset local position to 0,0,0 origin.	1.0
MOVE	8	int	Translate (+) forward or (-) back mm distance at SETV speed	1.0
ROTATE	9	int	Rotate (+) counter- or (-) clockwise degrees/sec.	1.0
SETRV	10	int	Sets maximum/turn rotation velocity; degrees/sec.	1.0
VEL	11	int	Translate at mm/sec forward (+) or backward (-) (SETV limit).	1.0
HEAD	12	int	Turn at SETRV speed to absolute heading; ±degrees (+ = ccw).	1.0
DHEAD	13	int	Turn at SETRV speed relative to current heading; (+) counter- or (-) clockwise degrees.	1.0
SAY	15	str	Play up to 20 pairs of duration, tone sound pairs through User Control Panel piezo speaker.	1.0
CONFIG	18	none	Request a configuration SIP.	1.0
ENCODER	19	int	Request one (1), a continuous stream (>1), or stop (0) encoder SIPs.	1.0
RVEL	21	int	Rotate robot at (+) counter- or (-) clockwise; degrees/sec (SETRV limit).	1.0
DCHEAD	22	int	Adjust heading relative to last setpoint; ± degrees (+ = ccw)	1.0
SETRA	23	int	Change rotation de(-) or (+)acceleration, in degrees/sec <sup>2</sup>	1.0
SONAR	28	int	1=enable, 0=disable all the sonar; otherwise, use bit 0 to enable (1) or disable (0) a particular array 1-4, as specified in argument bits 1-4.	1.0
STOP	29	none	Stop the robot; motors remain enabled	1.0
DIGOUT	30	2 byte	Set (1) or reset (0) User Output ports. Bits 8-15 is a byte mask that selects, if set (1), the output port(s) for change; Bits 0-7 set (1) or reset (0) the selected port(s).	1.0
VEL2	32	2 byte	Set independent wheel velocities; bits 0-7 for right wheel, bits 8-15 for left wheel; in 20mm/sec increments.	1.1
GRIPPER	33	int	Gripper server commands. See the Gripper or PeopleBot Manual for details.	1.0
ADSEL	35	int	Selects the A/D port number for reporting ANPORT value in standard SIP.	1.0
GRIPPERVAL	36	int	Gripper server values. See Gripper or PeopleBot Manual for details.	1.0
GRIPREQUEST	37	none	Request one (1), a continuous stream (>1), or stop (0) Gripper SIPs.	1.0
IOREQUEST	40	none	Request one (1), a continuous stream (>1), or stop (0) IO SIPs.	1.0
TTY2	42	str	Sends string argument to serial device connected to AUX1 serial port.	1.0

GETAUX	43	uint	Request to retrieve 1-200 bytes from the AUX1 serial port; 0 flushes the buffer.	1.0
BUMP_STALL	44	int	Stall robot if no (0), only front (1) while moving forward, only rear (2) while moving backward, or either (3) bumpers contacted when robot moving in related direction.	1.0
TCM2	45	int	TCM2 module commands; see <i>TCM2 Manual</i> for details.	1.0
JOYDRIVE	47	int	1=allow joystick drive from port while connected with a client; 0 (default) disallows.	1.0
SONAR_CYCLE	48	uint	Change the sonar cycle time; in milliseconds.	1.0
HOSTBAUD	50	int	Change the HOST serial port baud rate to 0=9600, 1=19200, 2=38400, 3=57600, or 4=115200.	1.0
AUX1BAUD	51	int	Change the AUX1 serial port baud rate (see HOSTBAUD).	1.0
AUX2BAUD	52	int	Change the AUX2 serial port baud rate (see HOSTBAUD).	1.0
AUX3BAUD	53	int	Change the AUX3 serial port baud rate (see HOSTBAUD).	1.0
E_STOP	55	none	Emergency stop; very abrupt by overriding deceleration.	1.0
TTY4	60	str	Send string argument out to device connected at AUX3 serial port.	1.0
GETAUX3	61	int	Request to retrieve 1-200 bytes from the device connected at the AUX3 serial port; 0 flushes the buffer.	1.0
TTY3	66	str	Send string argument out to device connected at AUX2 serial port.	1.0
GETAUX2	67	int	Request to retrieve 1-200 bytes from the device connected at the AUX2 serial port; 0 flushes the buffer.	1.0
CHARGE	68	int	0=release; 1=deploy autocharging-docking mechanism.	1.0
ARM	70-80	int	Pioneer Arm-related commands. See Arm manual for details.	1.1
ROTKP	82	int	Change working rotation Proportional PID value.	1.0
ROTKV	83	int	Change working rotation Derivative PID value.	1.0
ROTKI	84	int	Change working rotation Integral PID value.	1.0
TRANSP	85	int	Change working translation Proportional PID value.	1.0
TRANSPV	86	int	Change working translation Derivative PID value.	1.0
TRANSPKI	87	int	Change working translation Integral PID value.	1.0
REVCOUNT	88	int	Change working differential encoder count.	1.1
DRIFTFACTOR	89	int	Change working drift factor.	1.0
SOUNDTOG	92	int	0=mute User Control piezo; 1 = enable.	1.0
TICKSMM	93	int	Change working encoder ticks per millimeter tire travel.	1.1
BATTEST	250	int	Artificially set the battery voltage; argument in tens volts (100=10V); 0 to revert to real voltage	1.0
RESET	253	none	Force a power on-like reset of the controller.	1.0
MAINTENANCE	255	none	Engage controller maintenance (ARSHstub) mode.	1.0

## ANNEXE 2 : Tableau des commandes implémentées dans MobileSim 0.2

NAME	CMD#	ARG	DESCRIPTION
PULSE	0		no-op
OPEN	1	int	start up standard devices and send SIP packets
CLOSE	2	int	stop standard devices
ENABLE	4	uint	enable (1) or disable (0) motors (note: however that motors can never be disabled in Stage, you can always drive)
SONAR	28	uint	disable (0) or re-enable (1) sonar
CONFIG	18		request a configuration packet
STOP	29	int	stops the robot from moving
VEL	11	int	set the translational velocity (mm/sec)
ROTATE	9	int	set rotational velocity, duplicate of RVEL (deg/sec)
RVEL	21	int	set rotational velocity, duplicate of ROTATE (deg/sec)
VEL2	32	2bytes	independent wheel velocities, first byte (signed) is right, second=left
HEAD	12	uint	turn to absolute heading 0-359 (degrees)
SETO	7	none	resets robots odometry back to 0, 0, 0
DHEAD	13	int	turn relative to current heading (degrees)
SETV	6	int	sets maximum velocity and MOVE velocity (mm/sec)
SETRV	10	int	sets the maximum rotational and HEAD velocity (deg/sec)
SETA	5	int	sets translational accelleration (mm/sec/sec)
SETRA	23	int	sets rotational accel(+) or decel(-) (deg/sec)
MOVE	8	int	translational move (mm)