UNIVERSITE LIBRE DE BRUXELLES

FACULTE DES SCIENCES

DEPARTEMENT D'INFORMATIQUE

Apprentissage d'une Tâche de Contrôle pour un Robot Mobile en LEGO Mindstorms

Mémoire présenté en vue
de l'obtention du grade de
Licencié en Informatique
par Benjamin Haibe-Kains

Directeur de mémoire :
M. Gianluca Bontempi

Année Académique 2002 - 2003

Je remercie mon promoteur G. Bontempi ainsi que toute l'équipe d'Arkanoid El'Turbo Animal, à savoir S. Collette, J. Hubert, A. Paszukiewicz et D. Muquardt.

Le soutien de ma famille et amis fut important dans la réalisation de ce mémoire.

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Chapitre 1

Introduction

Depuis les années 1960 [78, 51], la robotique a connu un essor considérable. Les progrès techniques ont permis la construction de robots de plus en plus perfectionnés : de nombreux périphériques, une puissance de calcul accrue et des moyens de communication performants.

Alors que les humains effectuent une grande variété de tâches, la fonctionnalité des robots est limitée à la tâche pour laquelle ils sont conçus et programmés. Les robots perçoivent leur environnement au travers de senseurs, permettant de percevoir le monde réel, et agissent sur celui-ci au travers d'actuateurs. Actuellement, les robots humanoïdes polyvalents sont loin d'être au point (bien que ces dernières années aient vu des progrès importants en ce sens [49]) au contraire des robots spécialisés qui ne cessent de s'améliorer. Voici quelques exemples de tâches pour lesquelles les robots sont particulièrement utiles :

- Fabrication et manipulation de matériaux. Les chaînes de production font un usage massif de robots pour les tâches répétitives comme le transport d'objets, la peinture ou l'assemblage.
- Assistance des êtres humains dans des tâches complexes telles que la chirurgie ou la conduite de véhicules pour des handicapés moteurs par exemple.
- Téléprésence. Transmission des perceptions sensorielles, principalement visuelles et tactiles, d'un manipulateur à distance à un opérateur humain, lui donnant ainsi l'impression d'être sur place.
- Réalité augmentée. Les capacités de l'être humain sont augmentées artificiellement grâce à des robots comme des exo-squelettes décuplant la force d'un opérateur humain.

Ces types de robots ne sont pas autonomes mais sont bien adaptés à la tâche spécifique qui leur est dévolue. La recherche concernant les robots non autonomes est déjà bien avancée contrairement à la robotique autonome qui en est à ses balbutiements.

Les enjeux de la recherche en robotique autonome sont importants. Deux domaines se démarquent par l'utilisation de robots autonomes :

- Commercial : il existe quelques plateformes de robotique autonome ayant réussi commercialement, comme le robot *AIBO* de Sony [29] ou les LEGO Mindstorms [27]. Il s'agit principalement de robots utilisés à des fins ludiques.
- Recherche pointue : certaines recherches très pointues utilisent des robots autonomes. Citons la mission *Mars Pathfinder* lancée par la NASA en 1996 [65] et l'étude des fonds sous-marins par le robot autonome *Sentry* [52]. L'exploration et l'accès à des lieux dangereux pour l'être humain, sont des domaines où la robotique autonome s'avère être une solution particulièrement prometteuse.

Les difficultés rencontrées par les robots autonomes sont liées à l'imprécision de leurs senseurs et actuateurs ainsi qu'à la complexité de l'environnement réel :

- Le monde réel est inaccessible. Les senseurs sont imparfaits et l'interprétation de leurs mesures est complexe.
- Le monde réel est non-déterministe du point de vue du robot. Les roues dérapent, les batteries se déchargent, ainsi il est impossible de savoir si une action pourra être réalisée.
- Le monde réel est non-stationnaire, les effets d'une action changent au cours du temps.

- Les interactions du robot doivent s'effectuer en temps-réel. Ses temps de réaction sont-ils suffisamment courts pour une interaction efficace ?
- Le monde réel est continu. Les états et les actions proviennent d'un continuum de configurations physiques, ce qui rend impossible leur énumération.

Le premier pas vers l'autonomie d'un être vivant est la coordination de ses mouvements et de ses perceptions, appelée relation sensori-motrice, qui donnera ensuite naissance à des comportements de plus en plus complexes. Dans un robot, une relation sensori-motrice lie les signaux des différents senseurs du robot à l'ensemble des commandes de ses différents actuateurs [61].

Illustrons la définition de la relation sensori-motrice dans un robot par l'exemple suivant : le but d'un robot mobile est de se déplacer du point A au point B. Pour ce faire, le robot pourrait disposer d'un système de positionnement *GPS*, d'une roue motrice et d'une roue directionnelle. Quelles seront les commandes moteurs à effectuer pour atteindre le point B ? D'une manière plus générale, quelle est la relation existante entre les commandes moteurs et le changement de position déterminé par le *GPS* ? Lors des premières expériences de ce type, la forme analytique de telles relations était étudiée par le concepteur qui l'implémentait ensuite sur le robot pour réaliser l'objectif considéré. Cependant, la complexité croissante des robots, au niveau des senseurs et des actuateurs par exemple, complique considérablement la tâche du concepteur.

Au début de la recherche en robotique autonome, l'architecture dite classique était utilisée. Dans cette architecture, les actions à effectuer sont sélectionnées par un traitement séquentiel de l'information sensorielle. Le traitement séquentiel est composé de modules tels que la perception, la prise de décision ou le contrôle des moteurs. Cependant, les premières expériences de contrôle de robots autonomes ont mis en évidence quelques difficultés inattendues. En effet, le contrôle tel qu'il était implémenté avec une architecture classique ne permettait pas au robot de faire face aux situations inattendues [78, 24]. C'est pourquoi, une approche alternative, basée sur le comportement (*behavior-based*) [4, 22, 23, 40], fut développée afin de répondre aux besoins d'autonomie d'un robot. Au contraire de l'architecture classique, le robot est décomposé en une hiérarchie de comportements distincts permettant la prise en compte du contexte global. De plus, cette architecture permet l'élaboration de comportements complexes tout en facilitant la tâche du concepteur.

L'utilisation de méthodes d'apprentissage est apparue dans les années 1980 [78]. Que ce soit pour l'approche classique ou comportementale, toute l'intelligence dans un robot provient de son concepteur. Le robot est lâché dans un environnement et agit selon la manière dont il a été programmé. Cette approche n'est pas nécessairement la meilleure, le concepteur n'ayant qu'une connaissance incomplète de l'environnement dans lequel le robot évolue. L'utilisation de méthodes d'apprentissage a été proposée comme solution afin que le robot acquiert les informations dont il a besoin. Les mesures sensorielles peuvent être utilisées non seulement pour agir, mais aussi pour améliorer les capacités d'action du robot dans le futur. Parmi les méthodes d'apprentissage, nous nous focaliserons sur l'utilisation de méthodes d'apprentissage local [7, 16, 82].

Le sujet du mémoire porte sur l'utilisation d'une méthode d'apprentissage local, le Lazy Learning [16, 19, 20, 18], pour le contrôle d'un robot mobile autonome. Il s'agira de choisir une plateforme matérielle et logicielle afin de construire ce robot mobile. Le développement et l'étude de la plateforme robotique sélectionnnée permettra son utilisation en vue d'une tâche de contrôle. Au regard de la plateforme robotique, la tâche de contrôle choisie est la recherche d'une source lumineuse par le robot mobile. Plusieurs politiques de contrôle seront mises en oeuvre à des fins de comparaisons théoriques et empiriques. Les différentes méthodes utilisées seront introduites pour ensuite être comparées sur base d'expériences sur le robot.

L'utilisation des LEGO Mindstorms permet de réaliser diverses expériences [59, 81, 48] et dans notre cas, de mettre en évidence la nécessité d'utiliser des techniques d'adaptation telles que les méthodes d'apprentissage [7, 78, 44, 20, 18, 61] pour le contrôle d'un robot mobile autonome.

1.1 Contributions du Mémoire

Lors du mémoire, nous avons développé une plateforme robotique, tant matérielle que logicielle. L'étude de cette plateforme ainsi que son développement ont permis la construction d'un robot

mobile autonome en LEGO Mindstorms et son utilisation dans le cadre de la réalisation d'une tâche de contrôle. Les contributions du mémoire sont d'ordre algorithmique, matériel, logiciel et expérimental.

1.1.1 Contributions Algorithmiques

Différents algorithmes sont mis au point afin de mener à bien ce mémoire :
- Protocole de communication entre la plateforme informatique et le robot (chapitre 4).
- Contrôle des moteurs grâce aux senseurs de rotation du robot (chapitre 4).
- Méthodes d'analyse statistique des senseurs de luminosité (chapitre 4) et des rotations du robot (chapitre 5).
- Algorithme pour la politique de contrôle basée sur l'analyse des données sensorielles (chapitre 5).
- Algorithme pour la politique de contrôle basée sur l'apprentissage de la relation sensori-motrice (chapitre 5).
- Algorithme pour la politique de contrôle adaptatif basée sur l'apprentissage de la relation sensori-motrice (chapitre 5).

1.1.2 Contributions Matérielles

La plateforme robotique LEGO Mindstorms est utilisée afin de construire un petit robot mobile autonome. La tâche de contrôle choisie a guidé le développement de la structure, de la cinématique ainsi que des senseurs du robot. Les contributions personnelles se situent donc au niveau du design du robot ainsi que des choix concernant les périphériques externes (chapitre 4).

De plus, une étude approfondie des senseurs est réalisée afin d'estimer le potentiel de la plateforme robotique LEGO Mindstorms (chapitre 4).

1.1.3 Contributions Logicielles

Une fois le robot construit, le développement de la partie logicielle de plateforme robotique LEGO Mindstorms permet son utilisation dans le cadre de la tâche de contrôle. Deux catégories de logiciels sont développés :

1. Gestion de matériel :
   - Contrôle des moteurs grâce aux senseurs de rotations sur le robot en C (chapitre 4).
   - Protocole de communication entre la plateforme informatique et le robot en C (chapitre 4).
   - Gestion des senseurs de luminosité et du multiplexeur actif sur le robot en C (chapitre 4).
2. Réalisation de la tâche de contrôle :
   - Analyse statistique des senseurs du robot en MATLAB [50] (chapitre 4).
   - Analyse statistique des rotations du robot en MATLAB (chapitre 5).
   - Validation du modèle linéaire et du Lazy Learning en MATLAB [15] (chapitre 6).
   - Politique de contrôle figée en C (chapitre 6).
   - Politique de contrôle basée sur l'analyse des données sensorielles en C (chapitre 6).
   - Politique de contrôle basée sur l'apprentissage de la relation sensori-motrice en C (chapitre 6).
   - Politique de contrôle adaptatif basée sur l'apprentissage de la relation sensori-motrice en C (chapitre 6).

1.1.4 Contributions Expérimentales

La construction du robot et le développement des logiciels d'exploitation de cette plateforme, permettent d'expérimenter, dans le monde réel, les différentes politiques de contrôle étudiées. Les expériences suivantes sont toutes présentées au chapitre 6 :

- Expérience sur la politique de contrôle figée.
- Expérience sur la politique de contrôle basée sur l'analyse des données sensorielles.
- Expérience sur la politique de contrôle basée sur l'apprentissage de la relation sensori-motrice.
- Expériences sur l'adaptation de la politique de contrôle basée sur l'apprentissage de la relation sensori-motrice. L'adaptation concerne l'ajout de nouvelles données en cours de contrôle et l'inversion des deux senseurs de luminosité.

Seule l'expérience permettant l'analyse des rotations du robot est présentée au chapitre 5.

1.2 Description des Chapitres

Chapitre 1 L'introduction présente le sujet du mémoire, à savoir la robotique autonome et son contrôle, ainsi que les contributions personnelles. De plus, un glossaire, une liste des notations, abréviations et acronymes utilisés sont donnés.

Chapitre 2 L'état de l'art présente un bref historique de la robotique autonome ainsi que l'état des connaissances des méthodes de contrôle dans ce domaine.

Chapitre 3 La plateforme robotique est décrite tant au niveau logiciel que matériel. Différentes plateformes robotiques sont présentées succinctement afin de justifier le choix de l'utilisation des LEGO Mindstorms.

Chapitre 4 L'utilisation de la plateforme robotique sélectionnée au chapitre 3, ainsi que les développements réalisés sont présentés et leurs apports expliqués. Il en résulte la construction d'un robot mobile autonome en LEGO Mindstorms.

Chapitre 5 Une fois la plateforme mise au point au chapitre 4, le point central du mémoire, à savoir la tâche de contrôle pour le robot mobile autonome, est présenté en termes théoriques. Différentes politiques de contrôle sont étudiées à titre de comparaison.

Chapitre 6 Un ensemble d'expériences sont effectuées afin d'illustrer les propos du chapitre précédent, en termes empiriques.

Chapitre 7 Ce chapitre présente la conclusion du mémoire ainsi que les futurs travaux possibles.

Annexe A Spécificités techniques du RCX.

Annexe B Code source du script MATLAB réalisant l'analyse statistique des techniques pour le senseur de luminosité.

Annexe C Pseudocode et code source du contrôle des moteurs du robot.

Annexe D Pseudocode et code source des différents éléments du protocole de communication.

Annexe E Figures supplémentaires pour l'analyse de rotations successives.

Annexe F Procédure et résultats des tests de permutation.

Annexe G Procédure et résultats de l'estimation par *bootstrap*.

Annexe H Pseudocode et code source des différentes techniques d'acquisition de données.

Annexe I Code source des scripts MATLAB pour la validation de modèle linéaire et non linéaire.

Annexe J Pseudocode et code source des différentes politiques de contrôle.

1.3 Glossaire

Actuateur Dispositif qui engendre par son action la force motrice nécessaire pour permettre au robot ou à sa structure mécanique de se mouvoir.

Agent Entité physique ou virtuelle possédant des ressources propres, capable de percevoir son environnement, d'agir sur lui et dont les comportements visent à satisfaire ses propres objectifs.

Byte-code Code écrit sous forme binaire, correspondant à une machine hypothétique dite virtuelle, issu de la compilation d'un fichier écrit en code source, et permettant son interprétation sur diverses plateformes pouvant émuler fonctionnellement la machine virtuelle.

Compilateur croisé Compilateur qui opère sur un processeur, tout en générant du code machine pour un ordinateur de type différent (*cross-compiler*).

Composant fonctionnel Module de traitement d'informations (exemple : perception ou contrôle des moteurs).

Démonstrateur Système expert de démonstration limité pour des raisons méthodologiques à un sous-ensemble représentatif du problème que l'on souhaite résoudre par la méthode des systèmes experts.

Environnement run-time Environnement dans lequel le programme est exécuté.

Envoi broadcast Envoi simultané des mêmes données à un très grand nombre de destinataires, sans effectuer aucune sélection parmi ceux-ci.

Firmware Microprogramme qui comprend un ensemble ordonné d'instructions et de données stockées d'une façon qui est fonctionnellement indépendante de la mémoire centrale.

Fonction *inline* Une fonction dite *inline* est une fonction dont le contenu est inséré dans le code à chaque appel. L'appel de la fonction n'est donc jamais réalisé, il n'est donc pas nécessaire d'enregistrer l'adresse de retour et les paramètres dans une pile.

Macro Instruction de prétraitement précédant la compilation, et consistant en général au remplacement d'une chaîne de caractères spécifique par une autre, souvent plus complexe, permettant ainsi de faciliter le développement.

Makefile Fichier comprenant une séquence de commandes automatisant la compilation.

Multiplexeur Dispositif électronique qui permet de grouper les signaux provenant de plusieurs sources en un seul signal composite, afin de les transmettre sur une seule voie de communication.

Open-source Code source que l'on rend disponible gratuitement pour qu'il puisse être modifié et redistribué, dans un contexte de développement communautaire.

Patinage Rotation, sans entraînement, des roues motrices d'un véhicule par suite d'une adhérence insuffisante.

Photodiode Détecteur photoélectrique constitué d'une diode à semi-conducteur qui produit un courant photoélectrique par absorption d'un rayonnement optique incident.

RCX Microcontrôleur LEGO Mindstorms.

Registre device Registre d'accès à un périphérique.

Senseur Dispositif sensible à une grandeur physique et permettant de la transformer en un signal de sortie mesurable (généralement un signal électrique).

Senseur passif Un senseur passif ne nécessite pas d'alimentation contrairement aux senseurs actifs.

1.4 Notations

C Fonction coût.

D_N Ensemble de N données observées.

f Facteur d'oubli pour la technique *RLS*.

G_N Erreur de généralisation d'un modèle.

k Instant de temps discret.

l_1 Mesure du senseur de luminosité gauche.

l_2 Mesure du senseur de luminosité droit.

L Unité de mesure de la luminosité. $L \in [0, 1024]$.

m_1 Moteur gauche.

m_2 Moteur droit.

S	Ensemble des états d'un système dynamique.
s	Statistique ou fonction de l'ensemble de données observées.
t	Instant de temps continu.
U	Ensemble des actions possibles sur un moteur.
u_1	Amplitude du mouvement pour le moteur gauche.
u_2	Amplitude du mouvement pour le moteur droit.
x_1	Mesure du senseur virtuel gauche.
x_2	Mesure du senseur virtuel droit.

Pseudocode. La notation utilisée est inspirée d'un site anglophone [31] fournissant un standard pour le pseudocode.

1.5 Abréviations et Acronymes

API	Application Programming Interface.
ARX	AutoRegressive with an input.
CRC	Cyclic Redundancy Check.
GPS	Global Positioning System.
I/O	Input/Output.
JVM	Java Virtual Machine.
LCD	Liquid Crystal Display.
LED	Light Emitting Diode.
LL	Lazy Learning.
LNP	LegOS Networking Protocol.
L-O-O	Leave-One-Out.
LS	Least Squares.
μLUNAR	Micro Lightweight Underlay Ad hoc Routing.
MSE	Mean-Squared Error.
NARX	Nonlinear AutoRegressive with an input.
PAR	Politique de contrôle basée sur l'Apprentissage de la Relation sensori-motrice.
PAS	Politique de contrôle basée sur l'Analyse des données Sensorielles.
PF	Politique de contrôle Figée.
RAM	Random Access Memory.
RIS	Robotics Invention System.
RLS	Recursive Least Squares.
RMSE	Root Mean-Squared Error.
ROM	Read Only Memory.

Chapitre 2

Architecture de Contrôle : Etat de l'Art

Le contrôle et les différents concepts associés abordés dans ce mémoire dépendent du type d'agent utilisé. Il sagit ici d'un robot mobile autonome. Il existe de nombreuses définitions concernant le concept de robot dont la suivante : agent actif et artificiel dont l'environnement est le monde physique [78]. La partie autonome du robot est définie par ses prises de décision propres basées sur ses informations sensorielles et ses actions sur le monde réel (figure 2.1). Le mécanisme

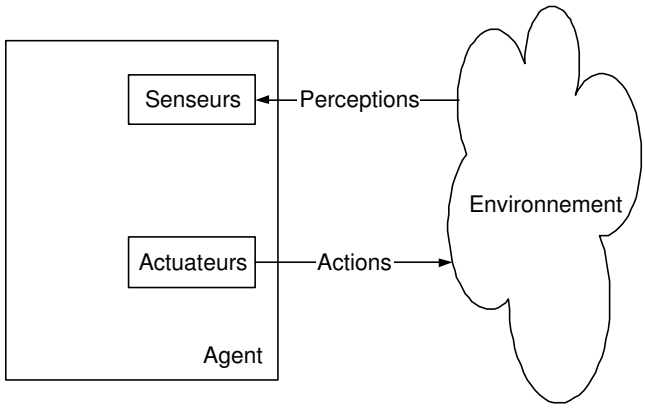


FIG. 2.1 – Modèle générique d'un agent.

de contrôle d'un robot est défini par son architecture qui spécifie comment la génération d'actions est organisée à partir des perceptions sensorielles [78]. Dans le cas des robots mobiles autonomes, l'environnement est le monde réel, donc complexe à gérer. De plus, bien que les informations sensorielles soient volumineuses et bruitées, le robot doit réagir rapidement dans certaines situations. Il est donc indispensable de mettre au point une architecture performante qui permette de remplir la tâche dévolue au robot en utilisant le matériel à disposition. Il n'existe pas, à ce jour, de théorie concernant le design architectural qui permette de prouver qu'un design est meilleur qu'un autre. C'est pourquoi, il existe plusieurs techniques utilisées selon la tâche et le robot.

2.1 Approche Classique

A la fin des années 1960, quelques outils primitifs pour robots intelligents étaient disponibles : des systèmes de vision pouvant localiser des objets simples, des outils de recherche de chemins

en deux dimensions, par exemple. Le design architectural de Shakey [51], premier robot mobile autonome (figure 2.2), fut conçu grâce à ces outils. L'implémentation de Shakey démontra l'importance de la recherche expérimentale en mettant en évidence quelques difficultés inattendues. Les chercheurs se confrontèrent à la difficulté de représenter le monde réel sous forme symbolique ou géométrique, et à l'inefficacité des *démonstrateurs* de théorèmes pour des plans non-triviaux.

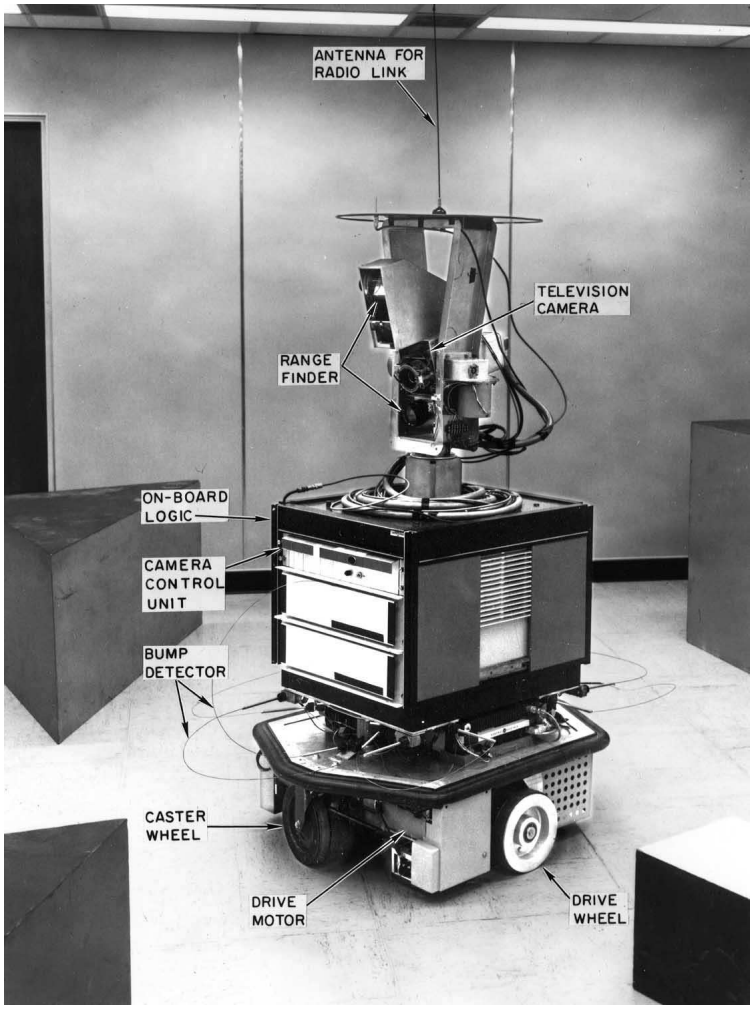


FIG. 2.2 – Robot construit en 1969 par l'institut de recherche de Stanford. Shakey était contrôlé par un ordinateur distant. Sa tâche était de se déplacer d'une position à l'autre en poussant des blocs.

L'approche classique en robotique consiste à surmonter les difficultés en construisant des robots les plus performants possible tant au niveau mécanique que sensoriel, ou en modifiant l'environnement [80]. Ces deux techniques sont souvent utilisées simultanément. Les différentes techniques de contrôle sont développées grâce à une connaissance approfondie du matériel et de l'environnement. Cette approche est systématiquement adoptée pour les robots industriels dont l'environnement est adapté à leurs tâches. L'ingénierie de l'environnement est aussi souvent introduit pour les robots mobiles autonomes dans les années 1960 [51] jusqu'à nos jours. Citons un exemple d'adaptation de l'environnement : des robots se déplaçant dans différents bureaux peuvent suivre une bande magnétique sur le sol afin de faciliter leur navigation. L'ingénierie minutieuse du robot et de son environnement augmente considérablement les coûts et ne peut s'appliquer à tous les types d'environnement.

La figure 2.3 représente la structure d'une architecture de contrôle classique. L'architecture

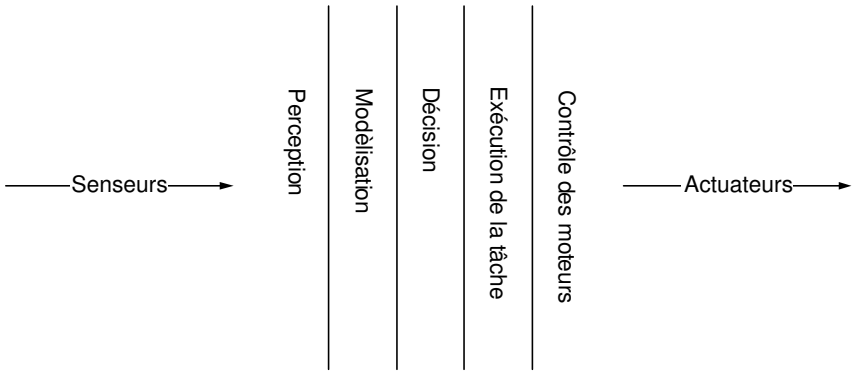


FIG. 2.3 – Décomposition traditionnelle d'un système de contrôle intelligent pour un robot mobile, c'est-à-dire une séquence de modules de traitement d'information (composants fonctionnels), des données sensorielles à l'action.

classique est une découpe verticale en *composants fonctionnels*, modules de traitement d'information des données sensorielles à l'action. Cette structure figée ne permet pas de faire face aux situations inattendues. Dès lors, la recherche en robotique a abordé le problème sous une autre approche dite *behavior-based*.

2.2 Approche Behavior-Based

En 1986, Rodney Brooks a recommandé une nouvelle approche appelée la robotique basée sur le comportement (*behavior-based robotics*) [24, 21, 4, 48]. L'idée est que l'entièreté du design du robot mobile autonome peut être décomposé, non pas en composants fonctionnels comme la perception, l'apprentissage et la prise de décision, mais en *comportements* tels que l'évitement d'obstacles, l'exploration ou le suivi de mur. Chaque module comportemental accède aux senseurs indépendemment afin d'en extraire juste l'information nécessaire, et envoie ses propres signaux aux actuateurs. Les comportements possèdent des niveaux de priorité et sont organisés en une hiérarchie dans laquelle les comportements haut-niveaux peuvent accéder à l'état interne des comportements bas-niveaux. Une telle hiérarchie est représentée par la figure 2.4.

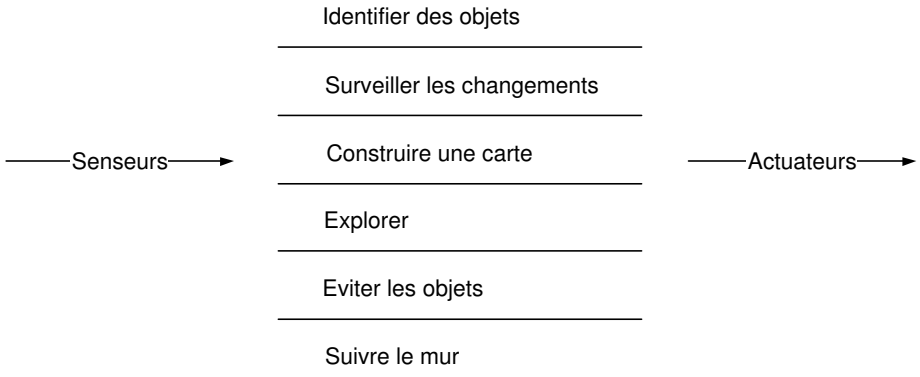


FIG. 2.4 – Design d'un robot mobile basé sur le comportement, représenté par sa décomposition en couches.

Le but principal de la robotique basée sur le comportement est d'éliminer la dépendance à une représentation complète et centralisée du monde réel, laquelle semble être l'aspect le plus coûteux de l'architecture de contrôle classique. L'état interne du robot n'est nécessaire que pour garder la trace des aspects du monde réel qui sont inaccessibles aux senseurs et qui sont requis pour la sélection des actions par chaque comportement. Pour les tâches pour lesquelles l'action appropriée est déterminée par les données sensorielles, le slogan "Le monde est son propre modèle" prend ici tout son sens. Par contre, dans certains cas, quelques bits de l'état interne sont suffisants même pour des tâches complexes.

Dans l'architecture classique, aucun des composants fonctionnels ne génère de comportement pour l'ensemble du système. Il est nécessaire de combiner un ensemble de composants fonctionnels afin d'obtenir un comportement. L'amélioration de la compétence du système passe donc obligatoirement par l'amélioration individuelle des composants fonctionnels. Au contraire, les modules de l'architecture moderne génère chacun un comportement. Il en découle que l'amélioration de la compétence du système passe par l'ajout de nouveaux modules comportementaux.

La tâche de contrôle étudiée dans le cadre de ce mémoire, s'inscrit dans une architecture *behavior-based* comme un comportement possible d'un robot mobile autonome.

2.3 Apprentissage

Dans un premier temps, toute l'intelligence dans un agent[1], provenait de son concepteur. L'agent est alors lâché dans un environnement et fait du mieux qu'il peut selon la manière dont il a été programmé. Cette approche n'est pas nécessairement la meilleure, tant pour l'agent que pour le concepteur. En effet, le concepteur n'a qu'une connaissance incomplète de l'environnement dans lequel l'agent évolue, l'apprentissage étant alors une voie possible pour que l'agent acquiert les informations dont il a besoin.

Les perceptions peuvent être utilisées non seulement pour agir mais aussi, pour améliorer les capacités d'action de l'agent dans le futur. L'apprentissage est le résultat de l'interaction entre l'agent et l'environnement, à travers l'observation de son propre processus de décision. Le domaine d'apprentissage est vaste, allant de la mémorisation triviale d'expériences à la modélisation de phénomènes. Un agent apprenant peut être divisé en quatre composants conceptuels représentés par la figure 2.5. Voici une description de chacun d'eux :

- Le composant de contrôle est responsable de la sélection des actions. Il considère les données sensorielles et décide des actions.
- Le composant d'apprentissage est responsable de l'apport d'améliorations. Il se base sur la rétroaction (*feedback*), c'est-à-dire l'observation du processus de décision de l'agent, afin de déterminer les modifications à effectuer au niveau du composant de contrôle pour obtenir de meilleurs résultats futurs.
- Le composant d'évaluation des performances permet de spécifier au composant d'apprentissage, la qualité des actions de contrôle effectuées. L'évaluation est rendue possible par la définition d'un standard de performance.
- Le composant de génération de problème est responsable de la suggestion d'expériences nouvelles apportant de l'information. Il permet une exploration des différentes actions possibles qui ne sont pas optimales à court terme mais peuvent s'avérer intéressantes à longue échéance.

Il existe trois grandes catégories de méthode d'apprentissage que sont l'apprentissage supervisé [10, 12, 44, 54, 73], l'apprentissage non-supervisé [41] et l'apprentissage par renforcement [78, 42, 35, 34, 33]. Elles se distinguent par le type de rétroaction disponible (voir figure 2.5). Dans le cadre de ce mémoire, nous nous focaliserons sur la méthode d'apprentissage supervisé pour laquelle la rétroaction est la suivante : l'agent prédit le résultat d'une action, effectue l'action et

[1]Nous introduirons dans cette section, les méthodes d'apprentissage en terme d'agent. Dans le cadre de ce mémoire, l'agent n'est autre que le robot mobile autonome construit en LEGO Mindstorms, mais le concept d'apprentissage ne se limite pas à ce type d'agent.

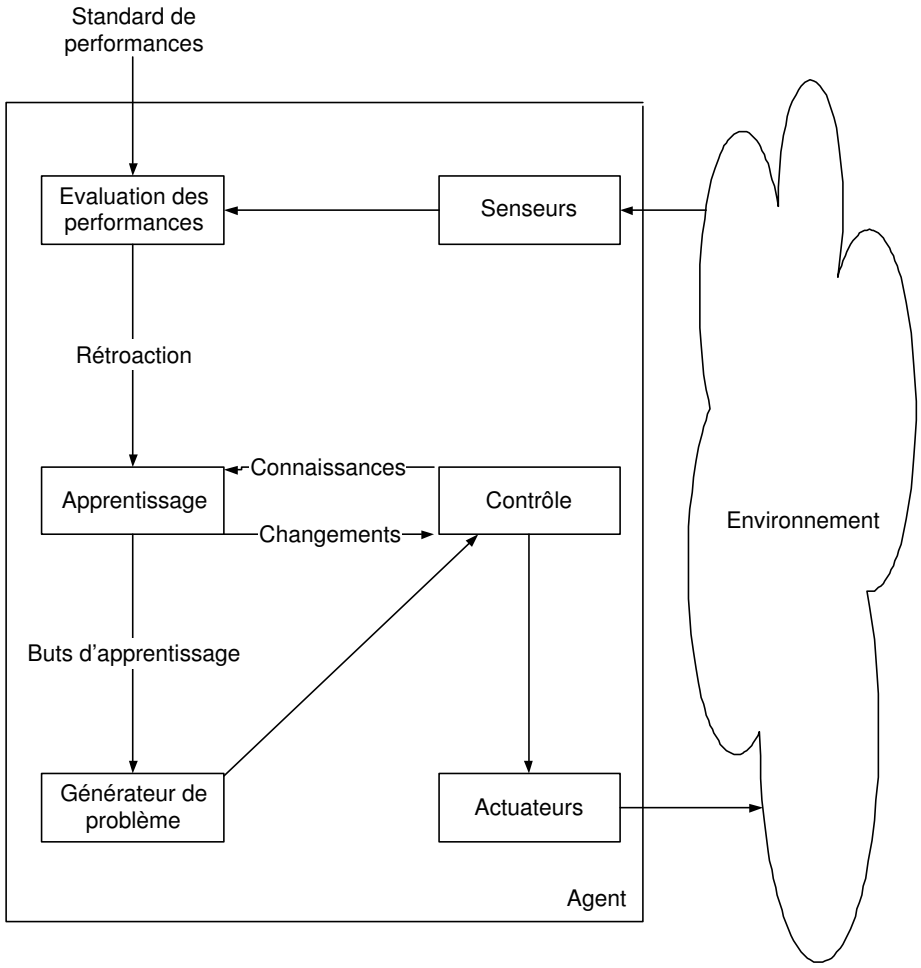


FIG. 2.5 – Modèle générique d'un agent apprenant.

l'environnement lui fournit directement, à travers ses senseurs, le résultat réel. La comparaison entre la prédiction et le résultat réel permet l'apprentissage de l'agent.

2.3.1 Méthode d'Apprentissage Local

La famille des modèles globaux est une famille de modèles traditionnellement utilisés pour la méthode d'apprentissage supervisé [16]. Les modèles globaux décrivent la relation entre les valeurs d'entrée et de sortie comme une seule fonction analytique sur l'ensemble du domaine d'entrée (voir figure 2.6). Parmi ces modèles paramétriques globaux, citons les modèles linéaires [36] et les

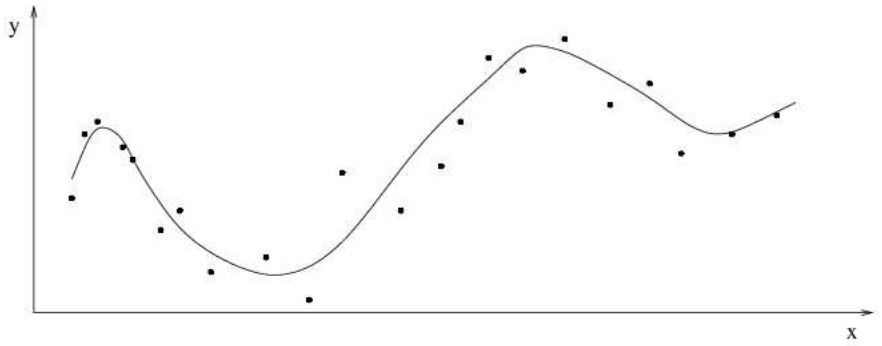


FIG. 2.6 – Le modèle global (ligne) qui est ajusté par l'ensemble des observations (petits points) pour un problème avec une variable entrée (axe des abscisses) et une variable sortie (axe des ordonnées). La figure est tirée de la thèse [16].

réseaux de neurones [77].

Une propriété intéressante des modèles globaux est le peu de mémoire requise pour la mémorisation du modèle, même pour un ensemble de données important. De plus, l'évaluation du modèle ne requiert que peu de calculs qui peuvent être exécutés en un temps réduit. Ces caractéristiques ont contribué au succès de l'approche globale.

Le problème de l'apprentissage d'un modèle paramétrique global à partir d'un ensemble de données de type entrée/sortie, peut être vu comme une estimation de fonction consistant à choisir, dans le domaine des fonctions paramétriques, celle qui approche le mieux la distribution de données inconnue. Malheureusement, la sélection de la fonction garantissant la meilleure faculté de généralisation, n'est pas triviale lorsque l'ensemble des données est limité.

C'est pour cette raison que des alternatives aux techniques de modélisations globales ont été développées comme l'approche *divide-and-conquer*. Son principe consiste à aborder un problème complexe en le divisant en problèmes plus simples dont les solutions peuvent être combinées afin d'obtenir une solution pour le problème original. Les problèmes plus simples peuvent être résolus par des techniques plus simples. Citons en statistique, les techniques linéaires développées au cours des années.

La modélisation locale [25] est une technique se basant sur l'approche *divide-and-conquer*. Elle considère le problème de l'estimation de la fonction comme un problème d'estimation de valeur. En effet, le but n'est pas de modéliser l'entièreté du phénomène stochastique mais de trouver la meilleure sortie pour une entrée de test, appelée ici une requête. Une fonction est approximée seulement pour le voisinage du point devant être prédit. Cette approche requiert la mémorisation de toutes les données d'entrée pour chaque prédiction contrairement au cas de la modélisation globale. Cependant, la modélisation locale ne requiert le calcul que d'approximateurs simples, par exemple constants et/ou linéaires, pour modéliser l'ensemble des données d'entrée au voisinage de la requête. Un exemple de modélisation linéaire local est donnée par la figure 2.7.

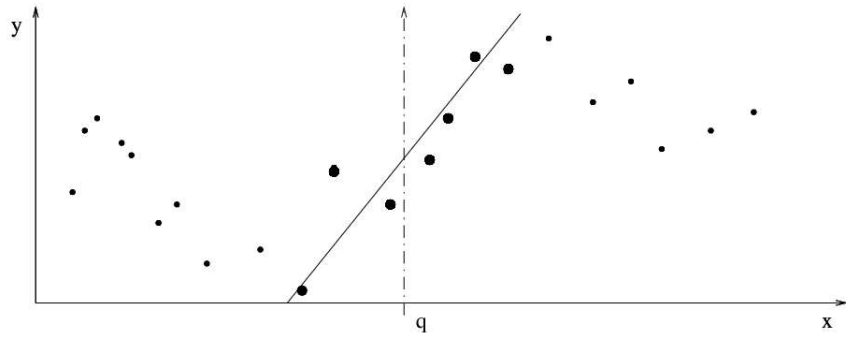


FIG. 2.7 – Modélisation locale d'une relation d'entrée/sortie entre la variable d'entrée x et la variable de sortie y, sur base d'un ensemble fini d'observations (petits points). La valeur de la variable y pour $x = q$ est trouvée par un modèle linéaire (ligne) qui est ajusté par les observations au voisinage (gros points) de la requête q. La figure est tirée de la thèse [16].

L'utilisation de méthodes locales pour le contrôle en robotique mobile autonome présente certains avantages :

- La modélisation de tâches complexes est plus facile. En effet, les méthodes d'apprentissage locales peuvent représenter des fonctions non-linéaires en calculant localement des approximateurs constants et/ou linéaires. Ces approximateurs sont simples à calculer.
- Le nombre d'observations à considérer peut être important. Les méthodes locales d'apprentissage gèrent aisément cette profusion d'observations et permettent un apprentissage rapide.
- Les flux d'informations provenant des senseurs du robot sont continus et la distribution des informations peut changer au cours du temps. Les méthodes d'apprentissage locales permettent une adaptation rapide aux nouvelles données.
- Les prédictions s'effectuent en temps-réel lors de l'arrivée d'une nouvelle requête.
- Les méthodes locales d'apprentissage évitent le problème complexe de la recherche d'une structure appropriée pour un modèle global mais utilisent des modèles locaux simples.
- Le but des méthodes locales d'apprentissage est d'améliorer les performances du contrôle aussi rapidement que possible, en utilisant le moins de données possible.

Des exemples d'expériences utilisant des méthodes d'apprentissage local ont montré leur efficacité dans le domaine du contrôle de robots autonomes [7, 82, 86, 63]. Un des exemples présenté dans l'article de C. Atkeson, A. W. Moore et S. Schaal [7], met en oeuvre un robot autonome jouant au billard (figure 2.8). L'équipement est constitué d'une queue actionnée par ressort avec un joint rotatoire contrôlé par un moteur, d'une table de billard et de deux caméras. Une des cameras est positionnée juste au-dessus de la queue et la deuxième filme l'ensemble de la scène afin de suivre les boules de billard. Une boule est placée aléatoirement en bout de table. Le robot frappe la boule se trouvant devant lui afin de mettre l'autre boule dans un des trous (figure 2.9).

L'implémentation de l'algorithme de contrôle utilise une méthode d'apprentissage local (*locally weighted regression* [7]). Afin de mener à bien la tâche de contrôle, il est nécessaire de disposer d'un modèle précis ce que peut fournir la méthode d'apprentissage local. En effet, après l'apprentissage, le robot montre un taux de réussite de 75%. Un second argument favorable à l'utilisation d'une méthode d'apprentissage local est la non-uniformité de la distribution de l'ensemble de données d'apprentissage due au processus d'exploration implicite. En effet, les actions produisant un mauvais coup ne seront que très peu utilisées ce qui explique que des performances élevées peuvent être obtenues avec un ensemble de données réduit. La méthode d'apprentissage local se focalise sur les coups donnant de bonnes performances.

FIG. 2.8 – Robot autonome frappant une boule de billard.

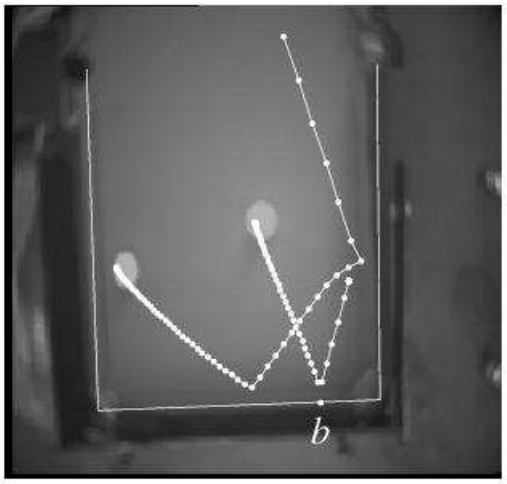


FIG. 2.9 – La trajectoire de chacune des deux boules est poursuivie en utilisant la caméra filmant la scène. Le point b indique la position de la collision entre la boule et le bord. Dans ce cas cas-ci le trou est manqué.

La méthode d'apprentissage local est un outil intéressant en robotique autonome comme en témoignent les remarques ci-dessous. Plusieurs expériences de ce type ont montré des performances prometteuses.

Chapitre 3

Plateforme Robotique

La plateforme robotique est constituée d'une plateforme matérielle et logicielle. Actuellement, aucune plateforme robotique n'est universelle. Il convient donc de choisir celle-ci en étudiant ses différents avantages et inconvénients.

Il existe une multitude de robots, chacun se distinguant par sa cinématique, sa puissance de calcul, ses périphériques embarqués, sa taille ou son autonomie par exemple. Dans le cadre de ce mémoire, quelques critères prévalent : le prix, la construction, la disponibilité, la fiabilité et les prérequis.

Parmi les plateformes disponibles, nous pouvons citer les MilliBots [56], les CotsBots [14], les Crickets [60], les Khepera [79] et les LEGO Mindstorms [27]. Toutes ces plateformes répondent aux critères cités ci-dessus excepté le prix et la disponibilité. En effet, seuls les LEGO Mindstorms sont adaptés à la réalisation de ce mémoire.

3.1 Plateforme Matérielle

Les LEGO Mindstorms sont une nouvelle génération de LEGO exploitant les possibilités de l'électronique et de l'informatique.

Cette plateforme est sélectionnée pour ses nombreux avantages :

- Cette plateforme robotique est très répandue notamment pour sa simplicité d'utilisation. Sa disponibilité ne pose pas de problème. De plus, le Département Informatique de la Faculté des Sciences de l'Université Libre de Bruxelles possède quelques kits qui ont pu être mis à contribution.
- Le prix est bas en comparaison aux autres plateformes robotiques. Le prix des quelques pièces qui ont du être commandées entraient dans le budget alloué au mémoire.
- La construction d'un robot mobile en LEGO Mindstorms ne requiert pas de connaissance en mécanique et électronique dépassant le cadre d'une licence en informatique.
- Le matériel est fiable et son remplacement facile.
- L'autonomie, bien que dépendante de l'utilisation du contrôleur et de ses périphériques, permet une utilisation intensive d'une à deux heures. Cette autonomie est plus que suffisante pour la réalisation de ce mémoire.
- Bien que limitée, la panoplie de senseurs et actuateurs disponibles pour les LEGO Mindstorms permet la réalisation de projets complexes. Outre les périphériques officiels, plusieurs sociétés [58, 70] en ont développé d'autres permettant d'augmenter considérablement le potentiel de cette plateforme robotique.

La plateforme LEGO Mindstorms est composée d'un kit de base appelé *RIS* (*robotics invention system*), d'un ensemble d'accessoires (*ultimate accessory set*), d'un ensemble de pièces spécialisées dans la construction de structures (*ultimate builders set*) et d'un kit de vision contenant une webcam (*vision command*).

3.1.1 Microcontrôleur

Le RCX est un microcontrôleur programmable inclus dans le kit de base *RIS*. Il fut développé conjointement par LEGO et le Massachusetts Institute of Technology [74] lors d'une étude de l'apprentissage de l'informatique aux enfants. Le MIT utilisait déjà des LEGO pour leurs prototypes (voir figure 3.1).

FIG. 3.1 – Brique programmable du MIT et brique LEGO RCX.

Il est constitué d'un microcontrôleur HITACHI H8/3292 [47] avec un processeur de type H8/300 cadencé à 16 MHz [46] ainsi que 32 Ko de *RAM*. Ce processeur permet de faire tourner un programme de contrôle qui accède aux périphériques intégrés du RCX. Parmi ces périphériques, citons les périphériques internes tels que les entrées/sorties au nombre de trois chacune, l'écran *LCD*, les boutons, le haut-parleur et l'émetteur/récepteur infra-rouge, et les périphériques externes que sont les senseurs et actuateurs décrits dans la section 3.1.2.

Une analyse succincte du RCX ainsi que les détails techniques sont donnés en annexe A. Une compréhension en profondeur du microcontrôleur est indispensable à l'écriture de programmes accédant aux périphériques comme nous le verrons dans la section 3.2.

La mémoire étant un élément clé dans le développement d'applications pour le RCX, la figure 3.2 montre la décomposition de la mémoire du RCX en plusieurs zones de fonction distincte.

Dans la *RAM* de 32 Ko, la mémoire programme totale vaut 28 Ko étant donné que 4 Ko sont réservés aux routines de gestions des périphériques RCX. Cette mémoire programme est partagée entre le système d'exploitation et les programmes utilisateurs. Le choix de la plateforme logicielle dépendra de l'espace mémoire disponible.

3.1.2 Périphériques Externes

Les périphériques externes pour la plateforme LEGO Mindstorms sont constitués des périphériques officiels et dérivés. La table 3.1 fournit une liste non-exhaustive des différents périphériques externes disponibles.

Bien qu'il existe de nombreux périphériques, le nombre d'entrée/sortie sur le RCX est faible ce qui limite la complexité des robots construits. L'utilisation de *multiplexeurs* améliore la situation mais cette limitation reste très contraignante.

3.2 Plateforme Logicielle

Deux parties distinctes composent la plateforme logicielle : le développement sur une machine hôte et le programme de contrôle tournant sur le RCX.

Type	Catégorie	Société	Description
Touché	Senseur passif	LEGO	Mesure binaire lors d'un contact.
Rotation	Senseur actif	LEGO	Mesure de rotation avec une précision angulaire de 22,5°.
Luminosité	Senseur actif	LEGO	Mesure de la luminosité entre 0 et 1024.
Vision	Senseur actif	LEGO	Webcam vendue par LEGO.
Moteur	Actuateur	LEGO	Moteur à vitesse variable.
LED	Actuateur	LEGO	Petite source lumineuse contrôlée électroniquement.
Compas électronique	Senseur actif	Public	Mesure du changement d'angle absolu. Seuls les plans électronique sont fournis sur Internet.
Détecteur de distance	Senseur actif	Mindsensors	Mesure de la distance entre 10 cm et 80 cm.
Température	Senseur actif	LogIT	Mesure de la température entre -30°C et 130°C avec une précision de 0,1°C.
Humidité	Senseur actif	LogIT	Mesure de l'humidité relative c'est-à-dire le pourcentage de vapeur d'eau contenue dans l'atmosphère à la température courante.
Son	Senseur actif	LogIT	Mesure du niveau sonore d'un son continu. La réponse du senseur aux sons courts est mauvaise.
Amplificateur pH	Senseur actif	LogIT	Mesure du pH entre 0 et 14.
Pression	Senseur actif	LogIT	Mesure de la pression absolue de l'atmosphère de 0 kPa à 200 kPa.
Mouvement et position	Senseur actif	LogIT	Potentiomètre calculant une rotation de 0° à 340°.
Voltage	Senseur actif	LogIT	Mesure de la différence de potentiel de -24 Volts à +24 Volts.
Multiplexeur passif	Multiplexeur	Mindsensors	Connexion de trois senseurs passifs.
Multiplexeur actif	Multiplexeur temporel	Mindsensors	Connexion de trois périphériques actifs.

TAB. 3.1 – Ensemble des périphériques externes pour le RCX.

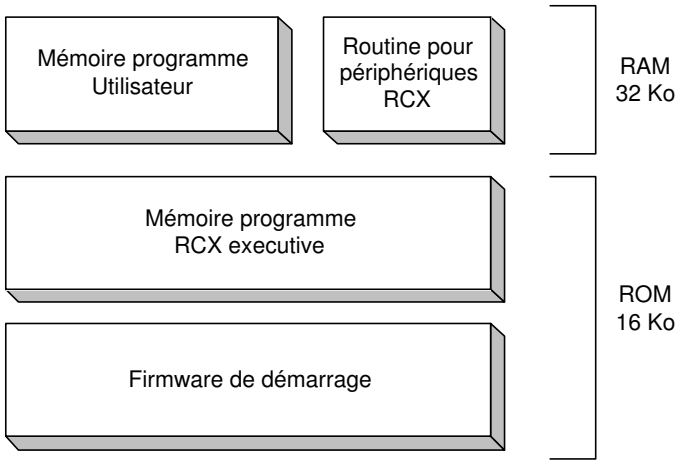


FIG. 3.2 – Décomposition de la mémoire du RCX, à savoir la *ROM* et la *RAM*, en zones de fonction distincte.

Le développement d'un programme de contrôle s'effectue sur la machine hôte. Le programme de contrôle est traité par un compilateur ou un assembleur croisé (*cross-compiler* ou *cross-assembler*) qui crée un exécutable en code natif pour le processeur du RCX. Cet exécutable est transféré au RCX par l'intermédiaire de la tour d'émission/réception infra-rouge du côté de la machine hôte ainsi que de l'émetteur/récepteur infra-rouge du côté du RCX. Cette opération est rendue possible par l'implémentation d'un protocole de transfert simple. Cette implémentation est exécutée par un programme, le RCX *executive*, résidant dans la *ROM* interne du RCX qui est lancé au démarrage ainsi qu'à la réinitialisation de celui-ci.

Le RCX *executive* sert aussi d'environnement run-time pour le programme de contrôle et fournit des services tels que les routines *I/O*.

Il existe de nombreux programmes de contrôle dont certains sont de véritables systèmes d'exploitation. Citons le logiciel de programmation LEGO Mindstorms, NQC [13], pbForth [45], leJOS [1] et brickOS [67]. Le logiciel LEGO Mindstorms présent dans le *RIS* fournit une interface de programmation simple sous forme de briques. Cette interface est accessible aux jeunes enfants mais ses possibilités sont très limitées pour une plateforme de robotique sérieuse. Le succès des LEGO Mindstorms a poussé des enthousiastes à développer des programmes de contrôle alternatifs.

3.2.1 Système d'Exploitation

L'utilisation du firmware délivré par LEGO pour le RCX n'est pas une solution envisageable à cause de ses limitations :

- Utilisation d'au plus 32 variables.
- Pas d'expressions.
- Pas d'appels de fonctions.
- Pas d'opérations en point flottant.
- Pas de routine d'accès au *LCD* ou aux boutons.
- Logiciel de développement uniquement disponible sur plateforme Microsoft Windows.

Les logiciels décrits ci-dessous, sont des alternatives intéressantes. Chacun d'eux possèdent des caractéristiques propres qui ont guidé le choix d'un système d'exploitation pour la réalisation de ce mémoire.

3.2.1.1 NQC

NQC (Not Quite C) [13] est un langage de programmation similaire au C qui fournit un contrôle plus souple que le langage graphique présent dans le *RIS*. Le plus grand avantage de NQC est d'utiliser le firmware de base ce qui évite la modification de celui-ci. Cependant, cette facilité implique que NQC est soumis aux mêmes contraintes.

3.2.1.2 pbForth

pbForth, pb signifiant *programmable brick*, est un système d'exploitation pour le RCX inspiré du Forth [64]. Le Forth est un langage de programmation adapté aux petits systèmes embarqués. Son utilisation dans le RCX permet d'avoir un système d'exploitation de 10 Ko, gérant ainsi 18 Ko de mémoire programme pour l'utilisateur. pbForth gère le multi-tâche coopératif. Cependant, les services fournis sont beaucoup plus limités que pour leJOS ou brickOS.

3.2.1.3 LeJOS

LeJOS est une version simplifiée du *Java 2 SDK* de Sun Microsystems [62]. Le compilateur *javac* produit un *byte-code* qui est interprété par la machine virtuelle java, appelée *JVM*, tournant sur le RCX. Cette machine virtuelle remplace le firmware initial du RCX.

LeJOS se distingue par une *API* étendue et bien documentée [2, 11]. La taille de la *JVM* est de 16 Ko ce qui laisse 12 Ko pour la mémoire programme. Son principal désavantage est la perte de performance engendrée par l'interprétation du *byte-code* au niveau du RCX.

3.2.1.4 BrickOS

BrickOS est construit autour du langage C et offre un bon compromis entre souplesse d'utilisation et efficacité. De plus, brickOS offre un large éventail de bibliothèques qui peuvent être incluses selon la volonté de l'utilisateur. La configuration de base de brickOS occupe 16 Ko de *RAM* du RCX mais les bibliothèques inutilisées peuvent être supprimées afin de gagner quelques précieux Ko. Toutefois, la configuration de brickOS est complexe.

L'architecture de brickOS est donnée par la figure 3.3 inspiré du mémoire [55]. Elle illustre l'introduction de la section 3.2 en utilisant brickOS. Notons que le nom original du projet était legOS et fut changé en brickOS durant l'année 2002.

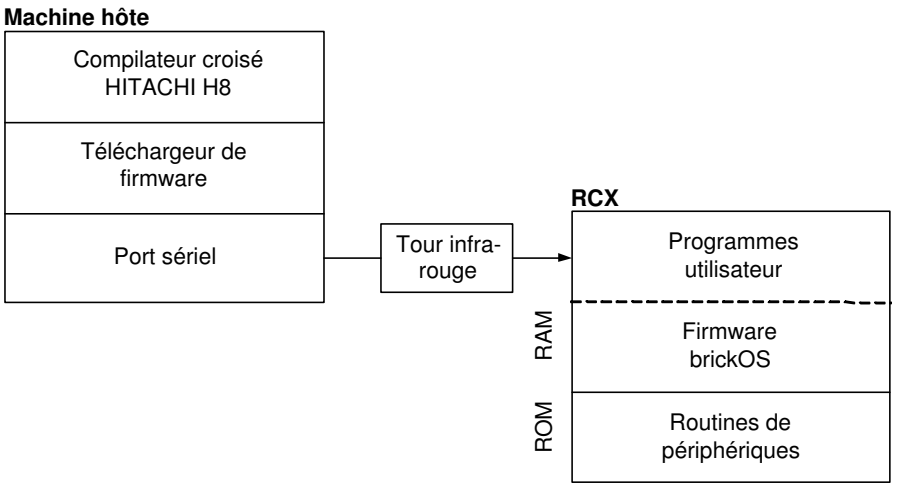


FIG. 3.3 – Architecture de brickOS.

BrickOS offre des services similaires aux systèmes d'exploitation. Voici quelques exemples des services offerts :

- Chargement dynamique des programmes et des modules.
- Gestion d'un protocole infra-rouge par paquet.
- Gestion dynamique de la mémoire.
- Multi-tâche préemptif.
- Assignation d'un niveau de priorité par tâche.
- Pilotes pour tous les périphériques RCX.

L'utilisation du compilateur croisé pour le processeur H3/800 et le *makefile* fourni par brickOS, est aisé pour les programmeurs travaillant habituellement sous Unix malgré une documentation pauvre [68, 66].

3.2.1.5 Analyse Comparative

La table 3.2 met en évidence les différentes caractéristiques déterminantes dans le choix de la plateforme logicielle. Elle est tirée du livre *CORE LEGO Mindstorms Programming* [9].

	Code RCX	**LeJOS**	**NQC**	**brickOS**	**pbForth**
Langage	Graphique	Java	Not Quite C	C et C++	Forth
Remplacement du firmware	Non disponible	Oui	Non	Oui	Oui
Nombres en point flottant	Non	Oui	Non	Non	Non
Affichage programmable	Non	Oui	Non	Oui	Oui
Trigonométrie et mathématiques avancées	Non	Oui	Non	Non	Non
Langage interprété	Oui	Oui	Oui	Non	Oui

TAB. 3.2 – Comparaison des différents langages de programmation pour le RCX.

Bien que sa configuration soit complexe et sa documentation pauvre [68], brickOS procure une souplesse d'utilisation et une efficacité supérieure. Le programme compilé en code natif atteint des performances plus élevées que celles obtenues par le *byte-code* java.

LeJOS fut d'abord utilisé pour sa complétude et sa documentation détaillée [2]. Une fois abordé le problème de la communication entre la machine hôte et le RCX, des tests de communication ont mis en évidence le manque de fiabilité du protocole de communication sous leJOS. L'interprétation du *byte-code* ne permettait pas une gestion performante de la communication. Le protocole de communication, expliqué dans la section 4.3, est mis au point sur brickOS car sa gestion du périphérique de communication infra-rouge est plus performante.

3.2.2 Protocole Réseau

Le protocole réseau de brickOS, appelé *LNP* (*LegOS Networking Protocol*) [5] permet une communication simple entre la machine hôte et le RCX. Ce protocole est constitué de quatre couches représentées par la table 3.3.

Les couches 0 et 1 ne présentent que peu d'intérêt pour le programmeur contrairement aux couches 2 et 3. Les couches 0 et 1 ne seront pas utilisées directement dans le cadre de ce mémoire. Les couches *integrity* et *addressing* sont présentées brièvement.

Integrity. La fonction *lnp_integrity_write()* envoie des messages *broadcast*. L'en-tête du paquet *integrity* ne contient pas de champ destination ou de champ source. Les paquets contiennent un

Couches	Nom	Description
3	Addressing	Identification des récepteurs et ports
2	Integrity	Paquets avec protection des blocs
1	Logical	Flux de bytes
0	Physical	Signal porteur modulé

TAB. 3.3 – Différentes couches du protocole *LNP*.

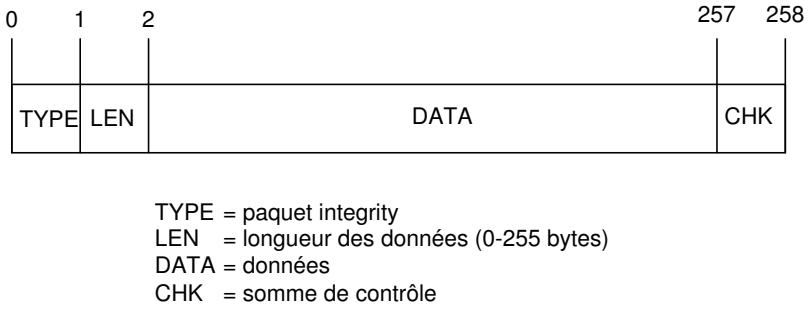


FIG. 3.4 – Format du paquet *integrity* dans la couche 2 du protocole réseau *LNP*.

maximum de 255 bytes d'information. La figure 3.4 représente le format du paquet et ces différents champs.

Au retour de la fonction, le paquet est envoyé sans erreur. En cas de collision, le paquet est automatiquement réémis. Aucune indication quant à sa réception n'est disponible.

Addressing. La fonction *lnp_addressing_write()* envoie des messages dont la destination et la source sont spécifiées dans les champs correspondants. Il s'agit en réalité d'un paquet *integrity* dans lequel sont encapsulées les informations d'adressage. La figure 3.5 représente le format du paquet et ses différents champs.

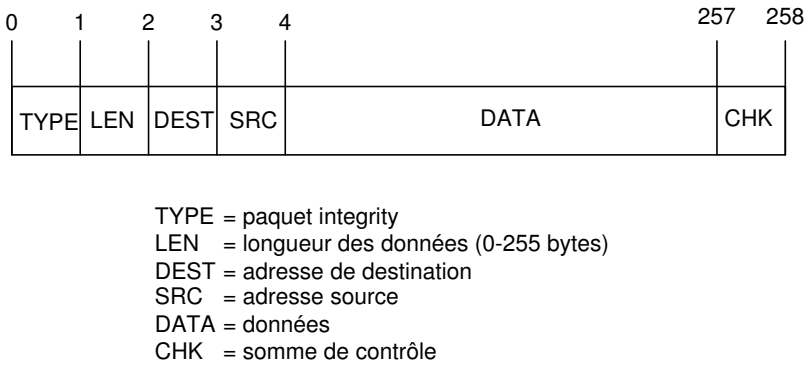


FIG. 3.5 – Format du paquet *addressing* dans la couche 3 du protocole réseau *LNP*.

Les quatre premiers bits du champ de destination contiennent l'adresse de destination et les quatre derniers bits fournissent le numéro de port choisi. La structure est similaire pour le champ SRC contenant l'adresse source. Il retourne de l'ajout de ces champs une augmentation de 40% de l'en-tête du paquet. Comme la fonction *lnp_integrity_write()*, aucune indication de réception n'est disponible.

Chapitre 4

Développement de la Plateforme

Ce chapitre décrit les choix effectués au niveau de la construction du robot ainsi que le développement des logiciels permettant l'exploitation de la plateforme robotique LEGO Mindstorms.

4.1 Robot

L'utilisation des LEGO Mindstorms permet une construction aisée d'un robot mobile. Cette facilité est un des avantages cités en section 3.1. La construction du robot est guidée par le choix de la cinématique ainsi que des périphériques utilisés.

4.1.1 Choix de la Cinématique

Le choix de la cinématique s'est porté sur le système de déplacement par poussée différentielle (*differential drive*). Il s'agit du plus simple et du plus répandu système de déplacement en robotique mobile.

Le système *differential drive* est constitué de deux roues propulsées chacune par un moteur indépendant ainsi que d'une roue libre assurant la stabilité du robot (figure 4.1).

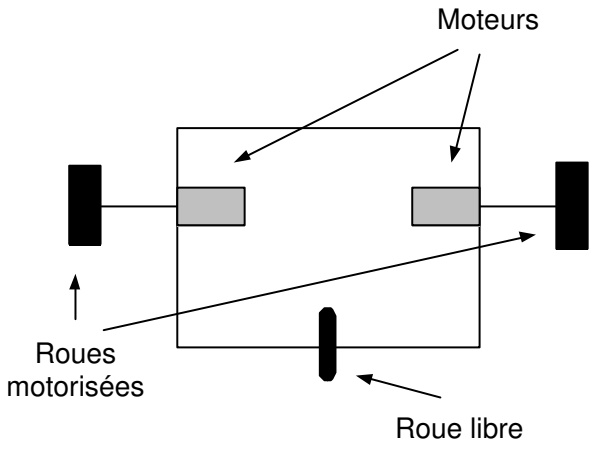
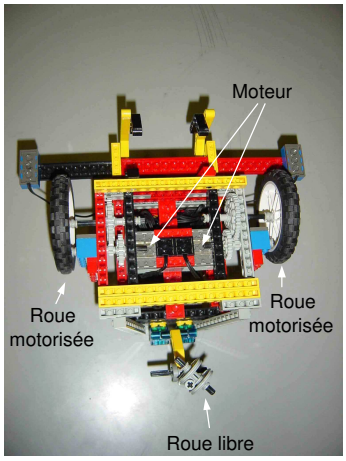


Schéma du robot mobile



Robot mobile construit en LEGO Mindstorms

FIG. 4.1 – Système de déplacement par poussée différentielle (*differential drive*). La figure de gauche montre un schéma du système tandis que la figure de droite montre le système sur le robot mobile construit.

L'avantage d'un tel système est sa simplicité de construction, les moteurs étant directement connectés aux roues. Néanmoins, la puissance des moteurs étant insuffisante pour faire avancer le robot à une allure acceptable, plusieurs engrenages sont interposés entre le moteur et la roue afin de démultiplier la puissance du moteur au détriment de la vitesse de rotation. Le principal désavantage est la difficulté qu'a le robot mobile de se déplacer en ligne droite. Ce type de mouvement est effectué en appliquant une même poussée par les deux moteurs. Or, ceux-ci ne sont pas exactement identiques ce qui a pour conséquence une déviation de la trajectoire du robot. De plus, la roue libre peut aussi influencer la trajectoire par frottement sur la surface de déplacement. Il n'existe pas de pièces LEGO permettant de construire une roue parfaitement libre. Les pièces ont été choisies pour leur frottement minime sur la surface de déplacement.

4.1.2 Choix des Périphériques

La tâche de contrôle mise en oeuvre dans le cadre de ce mémoire a guidé le choix des périphériques (voir section 3.1.2 pour une liste des périphériques externes disponibles) pour le robot mobile.

4.1.2.1 Moteur

Les moteurs utilisés pour la cinématique sont les moteurs de base livrés avec le *RIS* (figure 4.2). Ils permettent plusieurs vitesses de rotation[1] permettant ainsi un contrôle fin des moteurs.

FIG. 4.2 – Moteur de base livré avec le *RIS* de LEGO Mindstorms.

4.1.2.2 Senseur de Rotation

L'imprécision des moteurs LEGO Mindstorms et les difficultés inhérentes au choix de la cinématique ont suggéré l'utilisation de senseurs de rotation LEGO Mindstorms afin de gérer les mouvements du robot de manière plus évoluée qu'une gestion simple des deux moteurs. La technique utilisée sera détaillée en section 4.2. Le senseur de rotation (figure 4.3) permet de déterminer le sens de rotation ainsi que l'amplitude des rotations effectuées par une roue. Ce senseur permet de mesurer une rotation de $1/16^{\text{ème}}$ de tour, ce qui correspond à une précision de 22,5°.

Illustrons l'utilisation d'un senseur de rotation par un exemple simple : connectons un senseur de rotation directement sur la barre à laquelle est fixée la roue. Nous pourrions écrire un petit programme qui permettrait de bouger la roue d'un multiple de 22,5° afin d'obtenir des mouvements précis du robot. Un tour de la roue de 382,5° correspondrait donc à une incrémentation de 17 du compteur de rotations.

[1] L'utilisation de ces moteurs avec brickOS permet 256 vitesses de rotations différentes en marche avant et arrière.

FIG. 4.3 – Senseur de rotation LEGO Mindstorms.

4.1.2.3 Multiplexeur Actif

Nous avons vu qu'une des principales limitations du RCX est le nombre restreint d'entrées et donc de senseurs pouvant être connectés (voir section 3.1.1). Or, nous devrons connecter plus de trois senseurs actifs dans le cadre de ce mémoire. Une solution envisageable est l'utilisation d'un multiplexeur actif construit par la société Mindsensors [70] (voir figure 4.4). Celui-ci possède trois entrées, ce qui permettrait l'utilisation de neuf senseurs actifs dans le cas de l'utilisation de trois multiplexeurs actifs. Seul un exemplaire sera nécessaire dans notre cas.

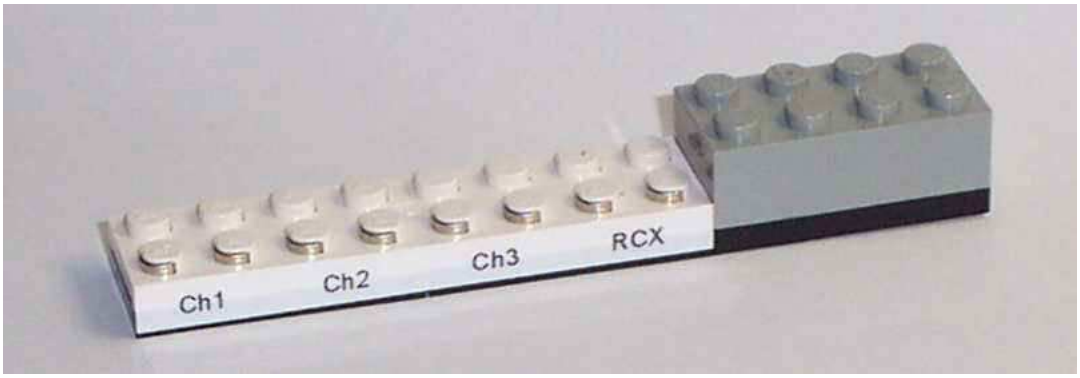


FIG. 4.4 – Multiplexeur actif. Le canal RCX est relié directement à une entrée du RCX. Les canaux labellisés 1, 2 et 3 peuvent accueillir chacun un senseur actif.

Les spécificités du multiplexeur [71] nous renseignent sur sa période d'échantillonage. Toutes les 10 ms, un senseur est échantillonné. Afin que l'ensemble des senseurs soient mis à jour, il ne faudrait théoriquement qu'un laps de temps de 30 ms. En pratique, les routines de gestion des périphériques sous brickOS ne sont pas appelées aussi fréquemment. Il en résulte qu'un laps de temps de 250 ms permet d'obtenir une marge de sécurité suffisante afin d'obtenir une mesure des senseurs mise à jour.

4.1.2.4 Senseur de Luminosité

L'utilisation première du senseur de luminosité LEGO Mindstorms est la mesure de l'intensité lumineuse entrant dans l'objectif devant le senseur (voir figure 4.5). Les objets sombres ont tendance à absorber plus de lumière ce qui implique qu'ils reflètent moins de lumière en retour au senseur de luminosité. Le senseur possède une petite *LED* infra-rouge qui illumine la scène devant lui et une photodiode qui mesure l'intensité lumineuse.

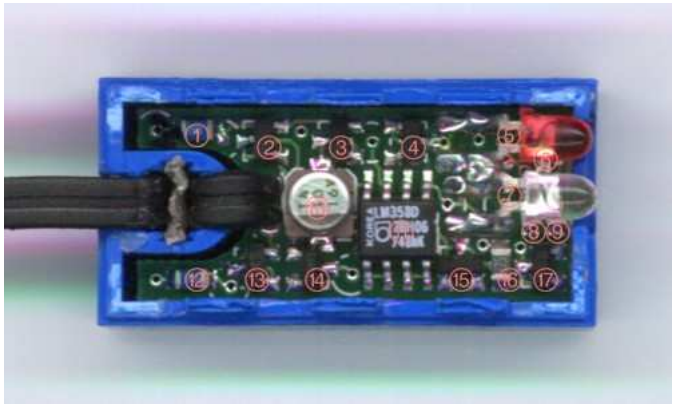


FIG. 4.5 – Senseur de luminosité LEGO Mindstorms. Devant le senseur, se trouvent la *LED* infrarouge et la photodiode.

BrickOS gère les senseurs de luminosité par des macros. Ces macros sont des fonctions *inline* effectuant une opération sur les valeurs brutes en sortie du convertisseur A/D auquel sont connectés les senseurs (voir annexe A pour les détails techniques). Une valeur entière de luminosité comprise entre 0 et 100 est renvoyée au programmeur. Cet intervalle de valeurs possibles est plus restreint que les valeurs brutes qui sont comprises entre 0 et 1024. L'utilisation directe des valeurs brutes permet d'obtenir une granularité de l'intensité lumineuse plus importante. La table 4.1 donne les correspondances entre le pourcentage d'intensité lumineuse et les valeurs brutes.

Valeurs brutes	Pourcentage de luminosité
225	-
322	100
450	82
565	65
785	34
945	11
1023	0

TAB. 4.1 – Tableau comparatif entre les valeurs brutes retournées par le senseur et son équivalent en pourcentages d'intensité lumineuse.

Dans le cadre de ce mémoire, les mesures de luminosité se baseront sur les valeurs brutes. Ces mesures seront donc comprises dans l'intervalle $[322, 1023]$, les valeurs extrêmes n'étant pas rencontrées en pratique[2].

Une expérience simple permet d'évaluer le potentiel du senseur de luminosité : le robot est placé juste en face de la source lumineuse et s'en éloigne progressivement jusqu'à atteindre une distance de six mètres. Chaque mesure est transmise à un ordinateur qui les stocke. Les mesures sont prises à chaque fois qu'un déplacement d'un centimètre est effectué. La première mesure effectuée s'avère donc être une valeur minimale et la dernière, une valeur maximale. En effet, une distance de six mètres entre la source lumineuse et le senseur suffit à ce qu'aucune information lumineuse ne soit enregistrée par le senseur. Le résultat est donné par la figure 4.6.

La relation entre la distance par rapport à la source lumineuse et l'intensité lumineuse n'est pas linéaire. Les petites variations de distance proche de la source lumineuse induisent des variations importantes de l'intensité lumineuse. Au contraire, loin de la source lumineuse (à partir de

[2]En général, nous observerons des mesures comprises dans l'intervalle $[350, 950]$.

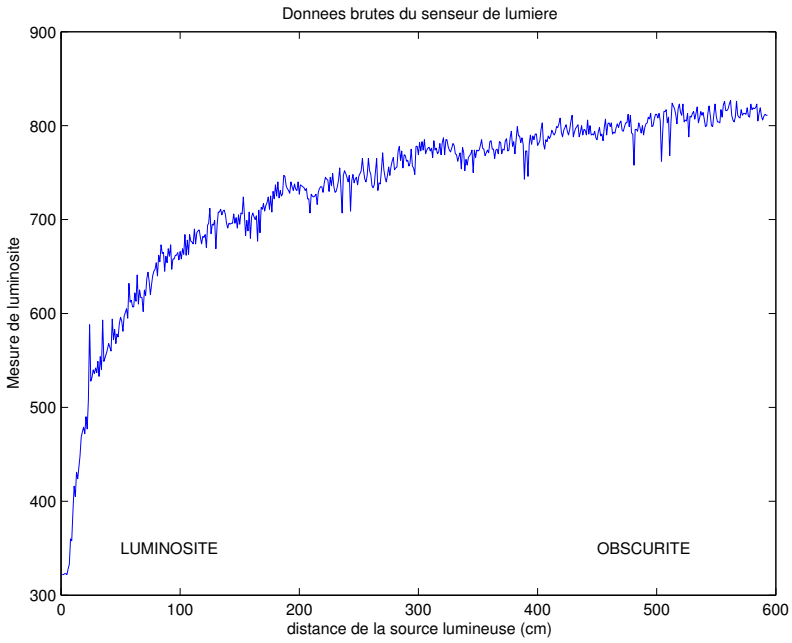


FIG. 4.6 – Senseur de luminosité.

cinq mètres environ), une grande variation de distance est nécessaire pour enregistrer une petite variation d'intensité lumineuse.

Nous pouvons voir sur la figure 4.6 que les mesures de luminosité ne sont pas sans à coups. La cause principale est la précision du senseur de luminosité LEGO Mindstorms. En effet, la variabilité des mesures est importante comme nous le montre l'analyse statistique suivante.

Analyse Statistique du Senseur de Luminosité

Trois techniques permettent de gérer différemment les senseurs de luminosité :

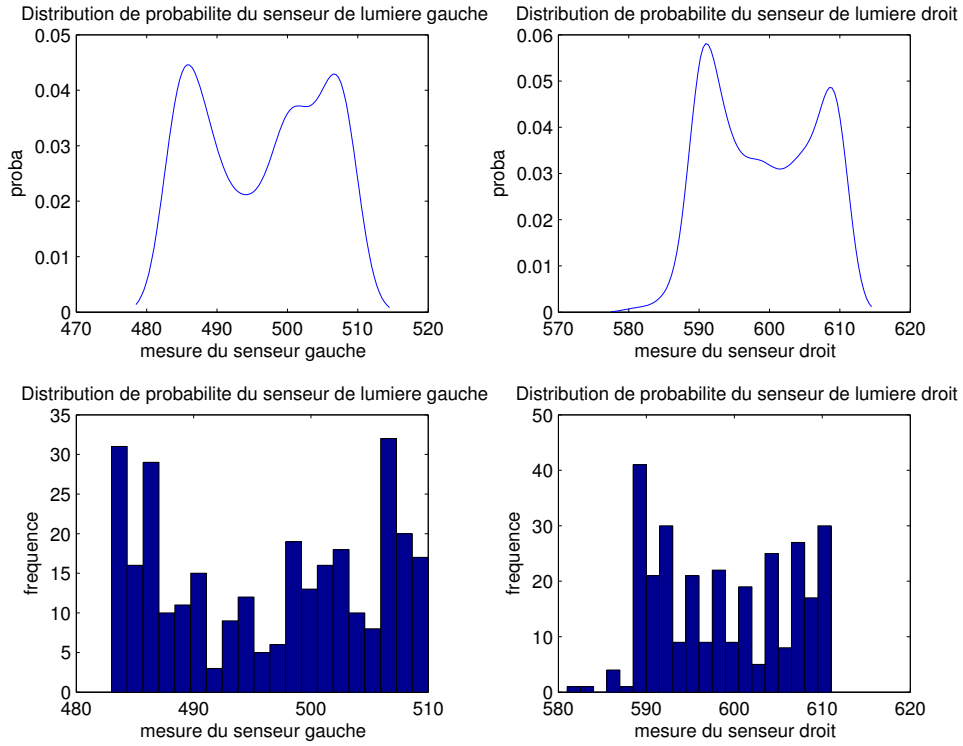
1. NOP (No OPeration) : une unique mesure est prise après le laps de temps nécessaire.
2. MEAN : une série de p mesures de luminosité sont prises et leur moyenne est calculée[3].
3. MEDIAN : une série de p mesures de luminosité sont prises et leur médiane est calculée.

Une série de 300 mesures de luminosité sont effectuées sans mouvement du robot avec les techniques NOP, MEAN et MEDIAN. Pour chaque technique, la variance est calculée. De plus, un histogramme et une estimation de la distribution de probabilité de chaque senseur sont tracés (voir figure 4.7, 4.8 et 4.9). Le code MATLAB [50] *comparison_sensors.m* est donné en annexe B.

La mise à jour par brickOS des mesures de luminosité à partir des registres *device* n'est pas exempte d'erreurs. Sporadiquement, une valeur hors-norme[4] est collectée après une action sur les moteurs. La technique NOP est sensible à cette erreur. L'utilisation de la technique MEAN permet de réduire la variance de l'erreur mais le calcul de la moyenne est sensible aux valeurs extrêmes [37]. Il en résulte que la technique MEDIAN est la plus efficace car le calcul de la médiane est moins sensible aux valeurs extrêmes [37] et présente des caractéristiques statistiques intéressantes en terme de variance. Cependant, le temps nécessaire pour une mesure de la luminosité est plus important.

[3]Ce paramètre est fixé à 5 dans le cadre de ce mémoire.

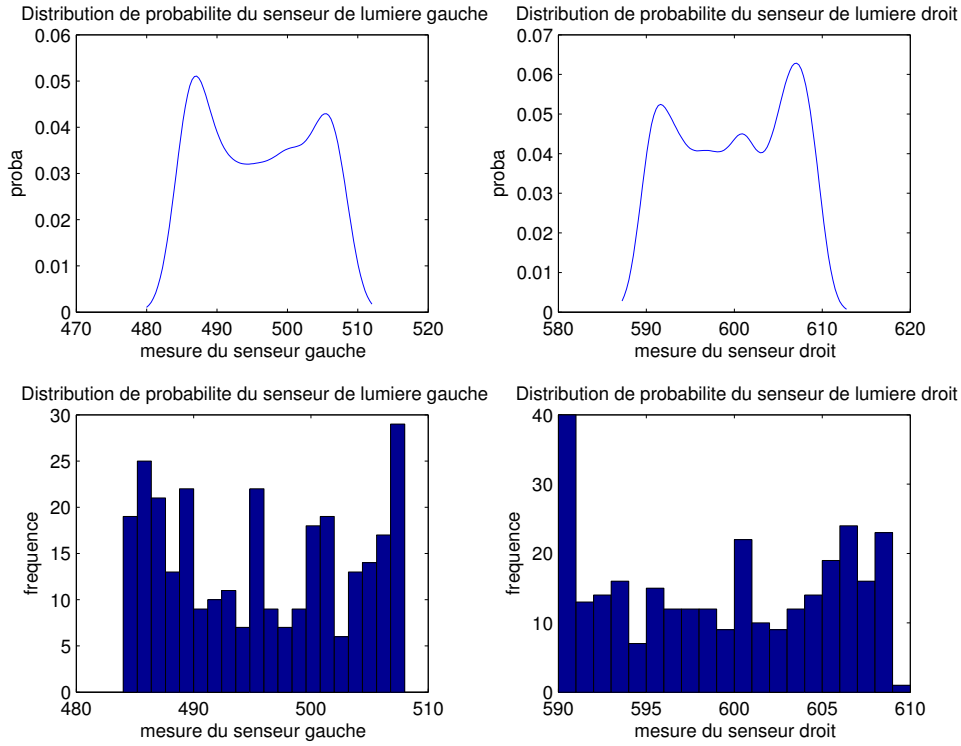
[4]La mesure de luminosité hors-norme est souvent une valeur supérieure à mille. Celle-ci ne correspond pas à l'intervalle de mesures observées lors d'une utilisation normale du senseur.



senseur gauche :
moyenne : 496.44
médiane : 498
variance : 77.2506

senseur droit :
moyenne : 599.06
médiane : 599
variance : 56.0499

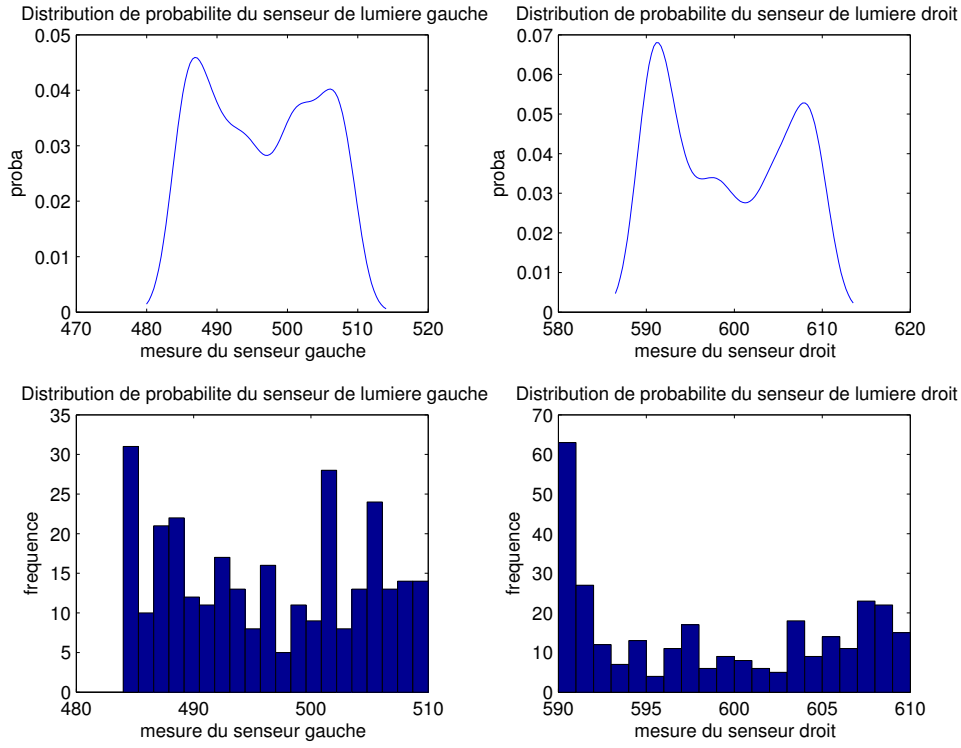
FIG. 4.7 – Technique NOP. Histogramme et estimation de la distribution de probabilité pour les deux senseurs de luminosité ainsi que les données statistiques correspondantes.



senseur gauche :
moyenne : 495.877
médiane : 496
variance : 59.0115

senseur droit :
moyenne :599.907
médiane : 600.5
variance : 39.3625

FIG. 4.8 – Technique MEAN. Histogramme et estimation de la distribution de probabilité pour les deux senseurs de luminosité ainsi que les données statistiques correspondantes.



senseur gauche :
moyenne : 496.343
médiane : 496
variance : 64.8483

senseur droit :
moyenne : 599.15
médiane : 598
variance : 50.5226

FIG. 4.9 – Technique MEDIAN. Histogramme et estimation de la distribution de probabilité pour les deux senseurs de luminosité ainsi que les données statistiques correspondantes.

Le senseur de luminosité et sa gestion MEDIAN constitue un senseur *virtuel*. L'utilisation du senseur virtuel (voir figure 4.10) plutôt que du senseur réel (voir figure 4.11) permet de limiter, au niveau du robot, les erreurs et la variance des mesures. En d'autres termes, il s'agit d'un prétraitement des données sensorielles afin d'en améliorer la précision.

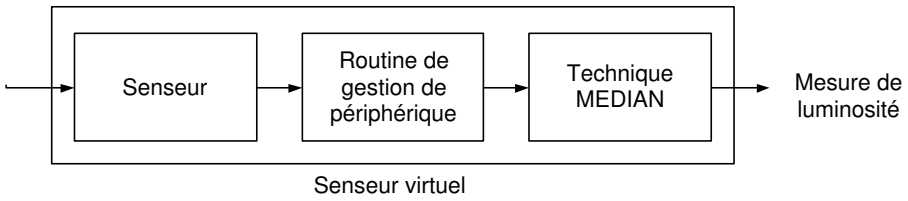


FIG. 4.10 – Senseur virtuel.

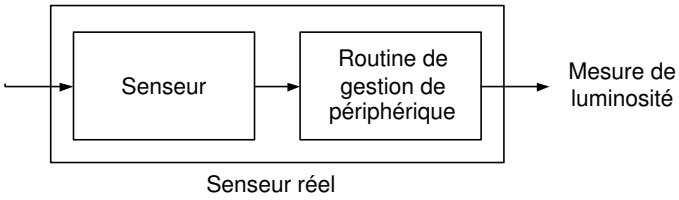


FIG. 4.11 – Senseur réel.

4.1.3 Représentation

La représentation du robot mobile permettra d'illustrer les concepts introduits tout au long du mémoire. Le schéma du robot est donné par la figure 4.12. Seuls les éléments importants sont représentés afin de ne pas alourdir inutilement le schéma.

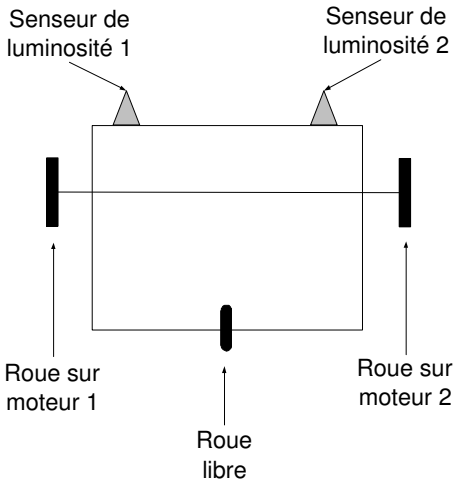


FIG. 4.12 – Schéma du robot et de ses principaux éléments.

La figure 4.13 montre le robot mobile construit en LEGO Mindstorms.

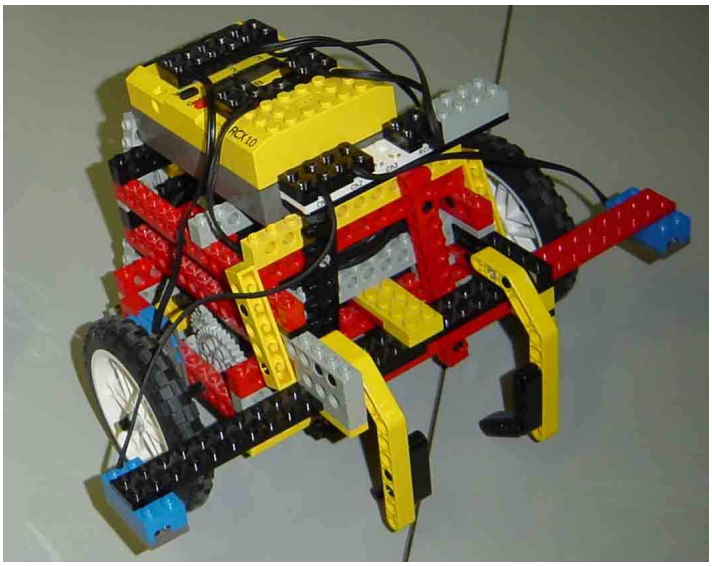


FIG. 4.13 – Robot mobile construit en LEGO Mindstorms.

4.2 Contrôle des Moteurs

Nous avons vu précédemment que nous ne pouvons obtenir du robot des mouvements de précision acceptable en n'utilisant que les moteurs gauche et droit. Afin de résoudre ce problème, un senseur de rotation est connecté à chaque moteur. Ces senseurs permettent de vérifier les rotations effectuées par chaque roue et ainsi de synchroniser celles-ci en temps réel. La synchronisation consiste à ralentir progressivement le moteur le plus rapide afin que les deux moteurs aient effectué le bon nombre de tours de roue. La technique est détaillée en annexe C.

La fonction *drive_ motors(int rotleft, int rotright)* permet au robot d'effectuer un déplacement en appliquant une amplitude[5] *rotleft* sur la roue gauche et une amplitude *rotright* sur la roue droite. La figure 4.14 montre quelques mouvements possibles en utilisant la fonction *drive_ motors()*.

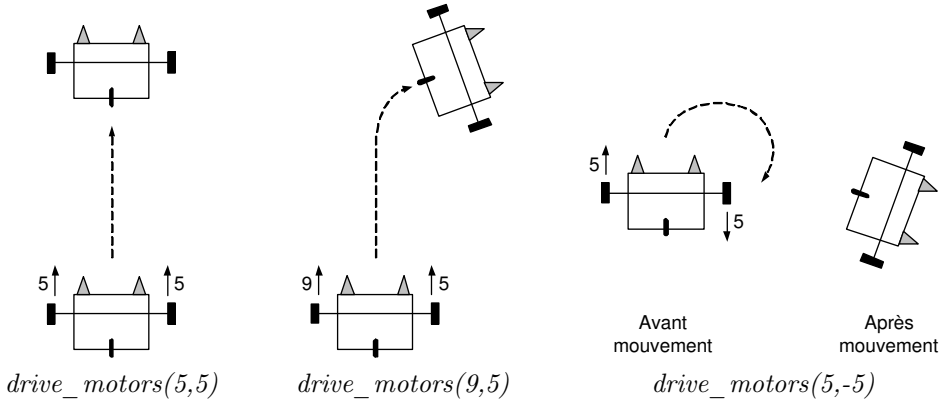


FIG. 4.14 – Exemples de mouvements possibles en utilisant la fonction *drive_ motors()*.

[5]L'amplitude d'une action sur une roue est un multiple de rotation complète de la roue. Par exemple, une amplitude de 2 vaut $2/40^{\text{ème}}$ de rotation complète de la roue considérée. Cette transformation permet d'avoir des valeurs d'entrées simples pour des petits mouvements.

De plus, une détection du blocage du robot est implémentée afin de déterminer si un mouvement effectué par le robot s'est correctement déroulé. Le senseur de rotation pouvant déterminer si une roue effectue une rotation, nous sommes en mesure de détecter un blocage du robot : si le robot est en train d'effectuer un mouvement et que les roues sont immobiles durant un laps de temps donné[6], l'action en cours est annulée et l'utilisateur est averti. Cette détection est inspirée d'une bibliothèque de contrôle développée par M. Patrick [72].

Bien que cette technique améliore la précision des mouvements du robot, il reste encore de nombreux facteurs d'incertitude tels que l'inertie, le type de surface ou le *patinage* des roues.

Le pseudocode de l'algorithme de contrôle des moteurs ainsi que son code source sont fournis en annexe C.

4.3 Protocole de Communication

Le développement d'un protocole de communication permettant le transfert bidirectionnel d'informations entre le RCX et la plateforme informatique, est indispensable dans le cadre de ce mémoire. En effet, l'utilisation de méthodes d'apprentissage pour le contrôle d'un robot mobile autonome nécessite une puissance de calcul et un espace mémoire dont le RCX ne dispose pas. Le processeur cadencé à 16 MHz et les 32 Ko de mémoire embarquée (voir section 3.1.1) ne suffisent pas à l'utilisation des techniques mises en oeuvre. Ce protocole permettra donc de limiter le travail du RCX à l'exécution d'ordres simples (action sur les moteurs ou prise de mesures par les senseurs par exemple), laissant ainsi le reste du travail à la plateforme informatique.

L'utilisation initiale de la communication infra-rouge entre une tour d'émission/réception et un RCX est le téléchargement des programmes destinés au RCX. Cependant, nous pouvons détourner l'utilisation première de la communication infra-rouge afin de communiquer avec le robot durant son activité. Néanmoins, la communication infra-rouge entre les tours d'émission/réception et les RCX est non fiable. En effet, bien que la portée de la communication infra-rouge soit suffisante (six à huit mètres), le RCX émettant/recevant des informations doit impérativement être dans l'axe de la tour émettant/recevant ces informations. Cette caractéristique rend obsolète l'utilisation d'une seule tour en présence de robots mobiles. En effet, une fois que la différence entre l'angle du robot et celui de la tour est trop importante, la communication devient impossible (figure 4.15). C'est pourquoi le développement d'un nouveau protocole de communication est nécessaire.

Le protocole de communication doit répondre aux besoins suivants :

- Emission et réception fiable des ordres et informations.
- Communication avec plusieurs RCX entre eux et avec la plateforme informatique.

Il existe plusieurs projets concernant le développement d'un protocole de communication évolué comme $\mu LUNAR$ [76] dont les objectifs et techniques utilisées sont différents. La mise en oeuvre du protocole de communication développé nécessite un ordinateur central et plusieurs ordinateurs clients auxquels sont connectés les tours d'émission/réception infra-rouges. Toute communication avec les robots s'effectue à partir de l'ordinateur central qui gère tous les ordinateurs clients. L'architecture est donnée par la figure 4.16. Afin de résoudre le problème d'angle entre une seule tour infra-rouge et un robot, plusieurs tours sont disposées autour de l'environnement expérimental. Cette disposition permet de communiquer avec un des robots à travers une ou plusieurs tours infra-rouges.

Un système d'accusés de réception permet au protocole de communication d'assurer une réception sûre des paquets envoyés. Si un paquet n'est pas convenablement reçu, celui-ci sera renvoyé autant de fois que nécessaire.

Le protocole de communication développé, appelé *tropocol*, se base sur le protocole réseau *LNP* (voir section 3.2.2). Il s'agit d'une quatrième couche similaire à la couche *adressing*. La table 4.2 montre les différentes couches du protocole de communication développé.

La couche *tropocol* introduit le format des deux types de paquets utilisés pour le protocole de communication (figure 4.17) : *ping* et paquet de données.

[6] Le délai est fixé à 300 ms.

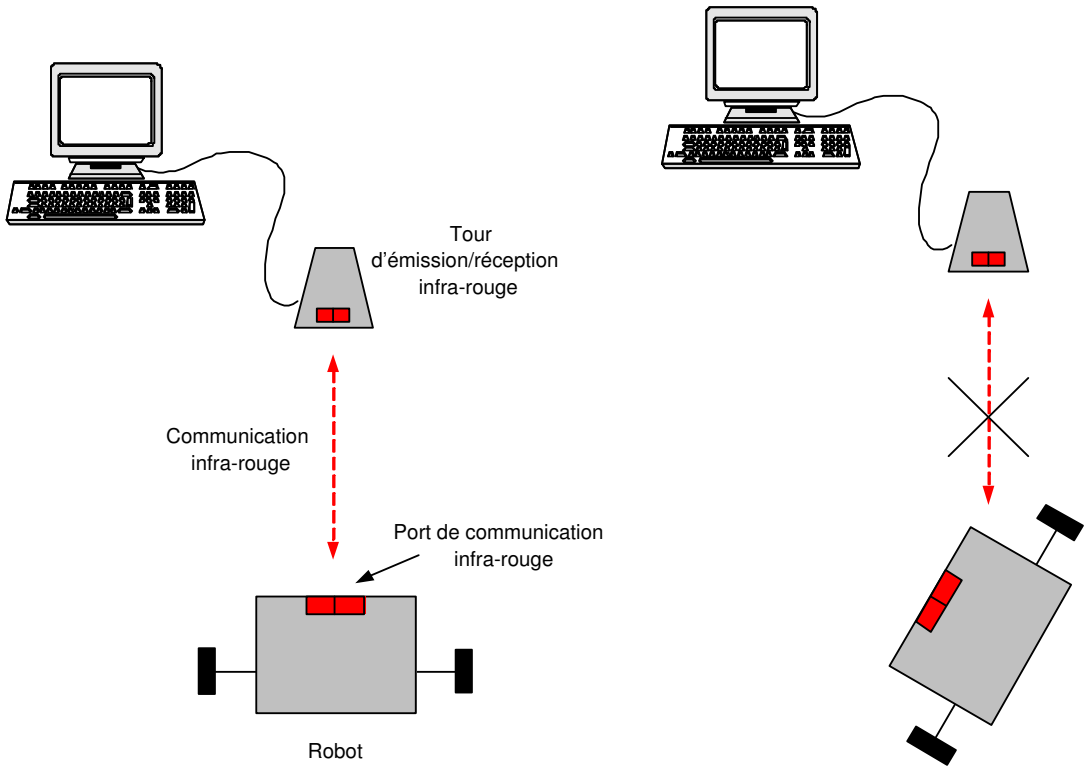


FIG. 4.15 – Communication infra-rouge entre la plateforme informatique et le robot. La différence d'angle entre le robot et la tour d'émission/réception infra-rouge détermine la fiabilité d'une communication.

Couches	Nom	Description
3	Tropocol	Identification des RCX et trames de données spécifiques
2	Integrity	Paquets avec protection des blocs
1	Logical	Flux de bytes
0	Physical	Signal porteur modulé

TAB. 4.2 – Différentes couches du protocole de communication *tropocol* dont les trois premières sont similaires au protocole *LNP*.

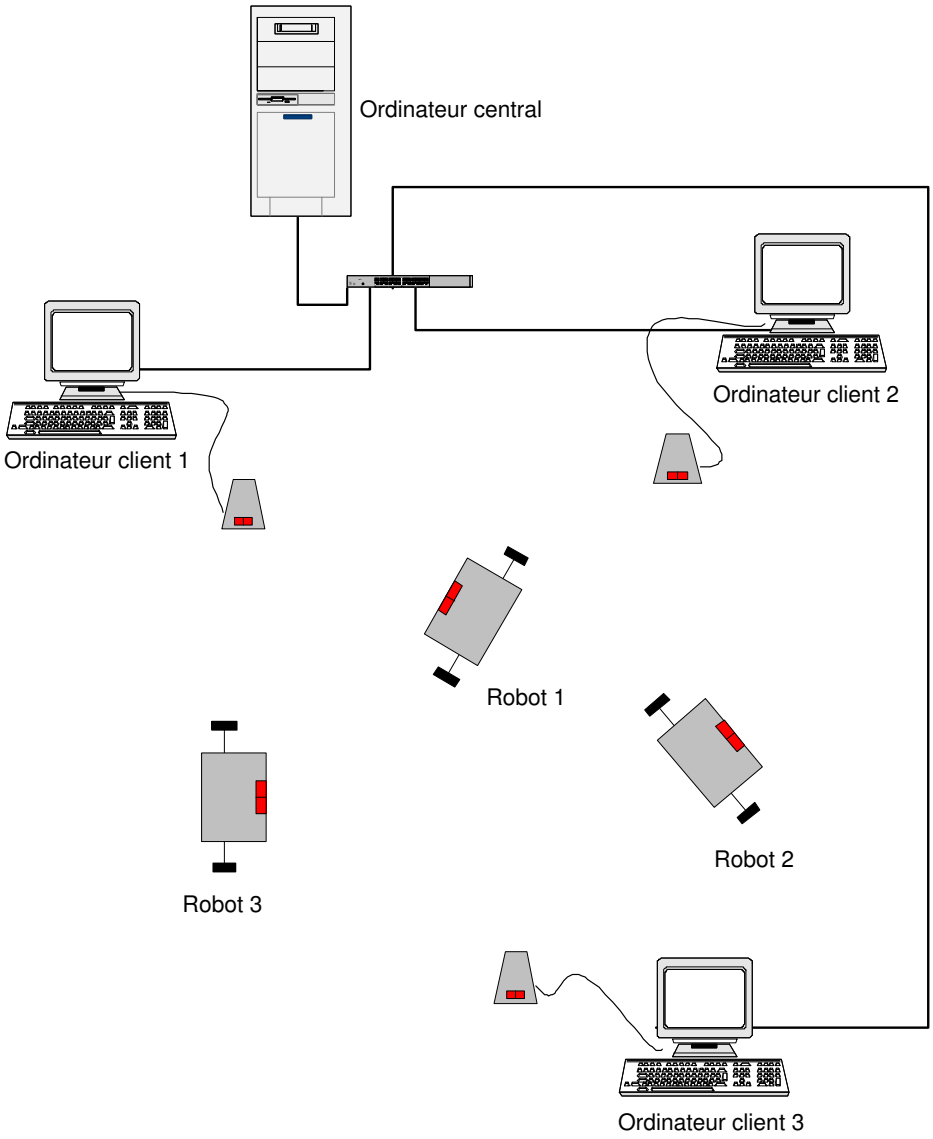


FIG. 4.16 – Architecture des différents composants nécessaires au protocole de communication développé.

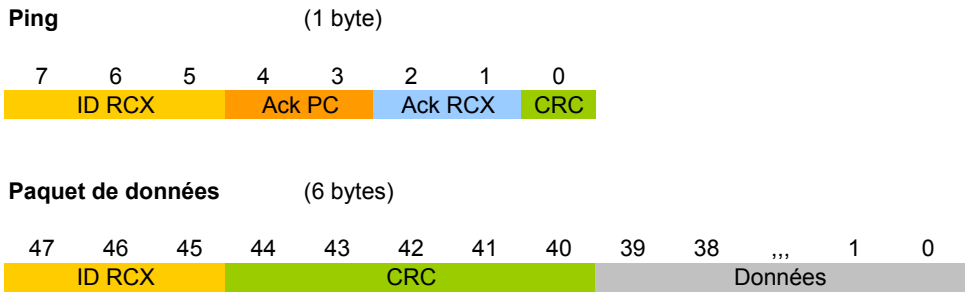


FIG. 4.17 – Deux types de paquets utilisés par le protocole de communication *tropocol*. Le paquet *ping* permet de localiser un robot spécifique. Le paquet de données permet l'envoi d'ordres au robot et l'envoi d'informations sur l'environnement ou l'état du robot.

Ping. Le paquet *ping* est envoyé à intervalle régulier par tous les robots en activité. Il permet la localisation des robots en terme de distance[7] aux tours d'émission/réception infra-rouges. Le champ *ID RCX* permet d'identifier le robot considéré. Les champs *Ack PC* et *Ack RCX* permettent la mise en place du système d'accusés de réception. Le champ *CRC* est un bit de parité permettant de vérifier l'intégrité d'un paquet *ping*.

Paquet de données. Le paquet de données est utilisé :
- Lorsque l'ordinateur central transmet un ordre à l'un des robots en activité.
- Lorsqu'un robot envoie des informations sur son environnement ou son état à l'ordinateur central ou un autre robot.

Le champ *ID RCX* permet d'identifier le robot considéré. Le champ *CRC* est un ensemble de cinq bits, chacun d'eux étant le bit de parité d'un byte de données correspondant. Par exemple, le deuxième bit du *CRC* (bit 41) est le bit de parité du deuxième byte de données (bits 8 à 16). L'intégrité d'un paquet de données peut ainsi être vérifiée par une technique requérant peu de puissance de calcul au niveau du RCX de chaque robot.

L'émission et la réception de paquets se déroule comme suit :
- Lorsqu'un robot émet un paquet, celui-ci est envoyé en *broadcast* et reçu par toutes les tours infra-rouges. Cependant, un seul exemplaire du paquet ne sera considéré par l'ordinateur central afin d'éviter les doublons.
- Lorsque l'ordinateur central émet un paquet vers un robot, un ordinateur client est choisi au préalable afin de n'émettre qu'à travers la tour la plus proche du robot considéré. A nouveau, l'envoi du paquet est effectué en *broadcast*, les robots ne considérant que les paquets qui leur sont destinés.

Le pseudocode et le code source des logiciels développés pour le RCX, les ordinateurs clients et l'ordinateur central sont fournis en annexe D.

Le protocole de communication fut développé en collaboration avec S. Collette, à l'occasion du Printemps des Sciences [32].

[7] La notion de distance entre un robot et une tour d'émission/réception infra-rouge est définie comme suit : une tour est proche lorsque la communication entre celle-ci et le robot est fiable. Sinon, la tour est dite lointaine.

Chapitre 5

Tâche de Contrôle

Lors du choix du mémoire, aucune plateforme robotique, tant matérielle que logicielle, n'était disponible. Afin de mettre sur pied cette plateforme ainsi que les outils nécessaires à son utilisation, nous avons considéré la mise en oeuvre d'une tâche simple, s'inscrivant dans un design comportemental pour un robot mobile (section 2.2) dont voici un exemple présenté par la figure 5.1.

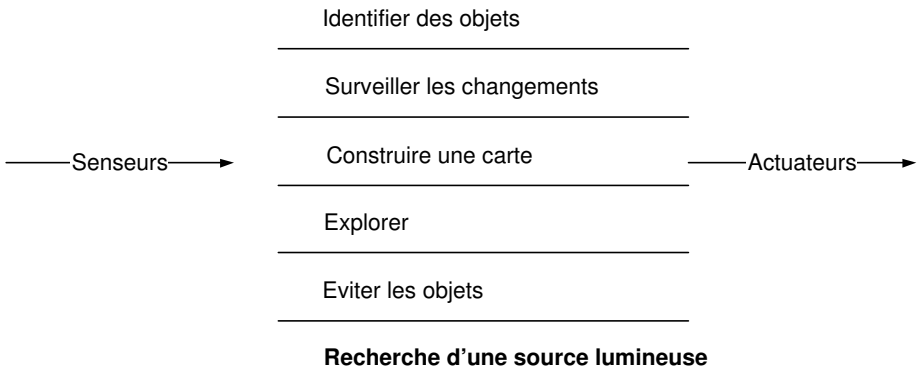


FIG. 5.1 – Design d'un robot mobile basé sur le comportement, représenté par sa décomposition en couches. La dernière couche est la tâche de contrôle étudiée dans le cadre de ce mémoire.

5.1 Description de la Tâche de Contrôle

La tâche de contrôle consiste en une recherche de source lumineuse. Pour ce faire, la cinématique du robot ainsi que ses senseurs de luminosité seront mis à contribution.

Comme nous avons pu le voir lors de la description de la construction du robot en section 4.1, deux senseurs de luminosité sont situés à l'avant de celui-ci. La tâche du robot mobile est de se diriger vers la source lumineuse comme le représente la figure 5.2.

Le robot n'a connaissance de son environnement qu'au travers ses senseurs de luminosité. Il n'agit sur son environnement qu'au travers sa cinématique et donc ses moteurs. Ses deux relations sont représentées par la figure 5.3.

Quelle que soit la politique de contrôle considérée, le déroulement est similaire. A des fins de simplification, le déroulement du contrôle se résumera à une succession d'étapes chacune composée de deux phases :

1. Phase de *positionnement* constituée d'une ou plusieurs actions de rotation. Une action de rotation est une action de type $[u_1,u_2]$ tels que $u_1 = -u_2$ où u_1 et u_2 sont des valeurs

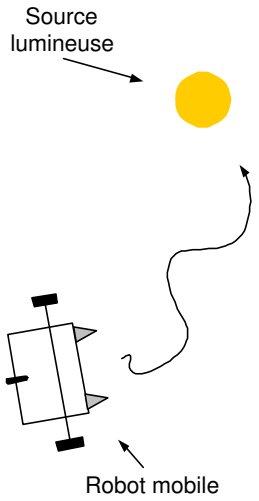


FIG. 5.2 – Recherche de la source lumineuse par le robot mobile. La source lumineuse est symbolisée par le cercle plein.

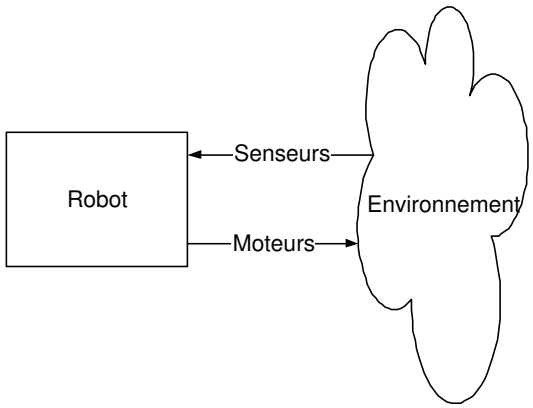


FIG. 5.3 – Connaissance et action du robot sur son environnement.

définissant l'amplitude d'une action appartenant à l'ensemble U des actions possibles sur les moteurs gauche et droit, notés respectivement m_1 et m_2. La phase de positionnement consiste à trouver la direction de la source lumineuse.

2. Phase de *rapprochement* constituée d'une action de déplacement de type $[u_1,u_2]$ tel que $u_1 = u_2$. Elle consiste à se rapprocher de la source lumineuse.

La figure 5.4 donne la représentation graphique utilisée pour une rotation et un déplacement. Ces représentations graphiques permettront d'illustrer le déroulement des différentes politiques de contrôle.

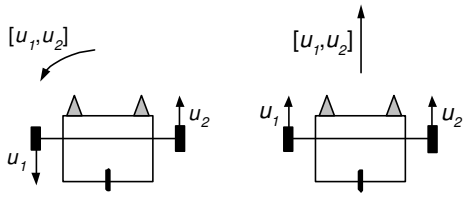


FIG. 5.4 – Représentation graphique d'une rotation et d'un déplacement.

La figure 5.5 représente la succession des deux phases et son résultat attendu. L'exécution répétée de ces deux phases permet de rapprocher le robot de la source lumineuse et ainsi, de mener à bien la tâche de contrôle.

La distance parcourue lors de la phase de rapprochement est fixée empiriquement. Le choix d'une telle distance est un compromis entre rapidité et performance. En effet, si une grande distance est parcourue, une erreur minime dans la recherche de la direction peut mener le robot à quitter l'axe vers la source lumineuse. Une petite distance permet de diminuer cette erreur mais augmente le nombre de phases de positionnement nécessaires à la réalisation de la tâche de contrôle. La figure 5.6 illustre le choix de cette distance.

Nous envisagerons, dans le cadre du mémoire, trois politiques de contrôle : politique de contrôle figée, politique de contrôle basée sur une analyse des données sensorielles et politique de contrôle basée sur l'apprentissage de la relation sensori-motrice.

5.2 Politique de Contrôle Figée

La construction du robot et l'étude préalable des senseurs nous fournissent une connaissance partielle du robot. Cette connaissance permet de réaliser la tâche grâce à une politique de contrôle écrite "à la main". La figure 5.7 représente la politique de contrôle figée.

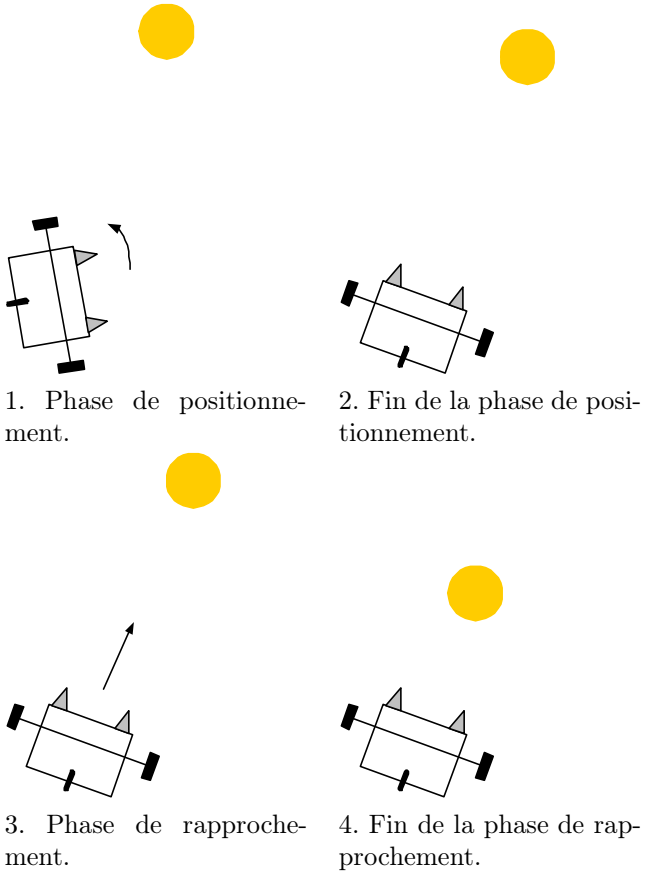
Cette politique de contrôle est implémentée à titre d'évaluation par rapport aux politiques de contrôle qui seront présentées en section 5.3 et 5.4.

5.2.1 Algorithme de Contrôle

L'algorithme de contrôle développé se base sur une exploration des directions du robot. Chaque étape se déroule en deux phases, l'une effectuant une série de rotations et l'autre avançant le robot dans la direction sélectionnée.

Une amplitude pour chaque rotation est fixée ce qui permettra de déterminer le nombre de rotations nécessaires pour un tour complet du robot sur lui-même. Les amplitudes des différentes actions utilisées dans l'algorithme sont déterminées empiriquement.

Durant la première étape, un tour complet est effectué par le robot. Le tour complet est nécessaire car la direction inconnue du robot dépend de son placement dans l'environnement expérimental. A chaque rotation, les mesures de luminosité ainsi que le numéro de la position considérée sont enregistrées. Le robot effectue ensuite une rotation d'amplitude suffisante pour



1. Phase de positionnement.

2. Fin de la phase de positionnement.

3. Phase de rapprochement.

4. Fin de la phase de rapprochement.

FIG. 5.5 – Déroulement d'une politique de contrôle constituée d'une phase de positionnement et d'une phase de rapprochement.

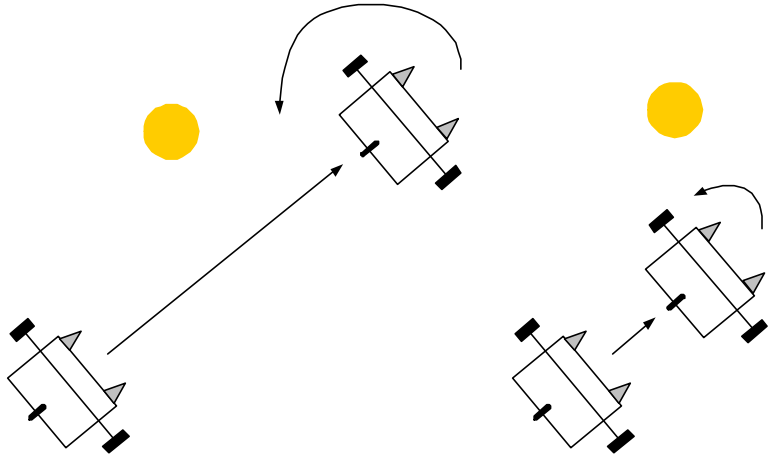


FIG. 5.6 – Choix de la distance parcourue lors de la phase de rapprochement.

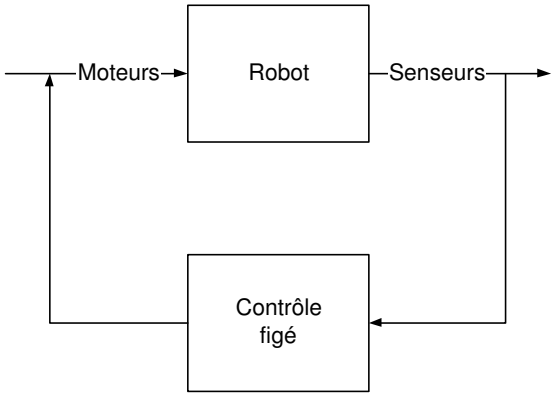


FIG. 5.7 – Politique de contrôle figée.

retourner à la position caractérisée par des mesures de luminosité maximale. La figure 5.8 illustre cette première étape.

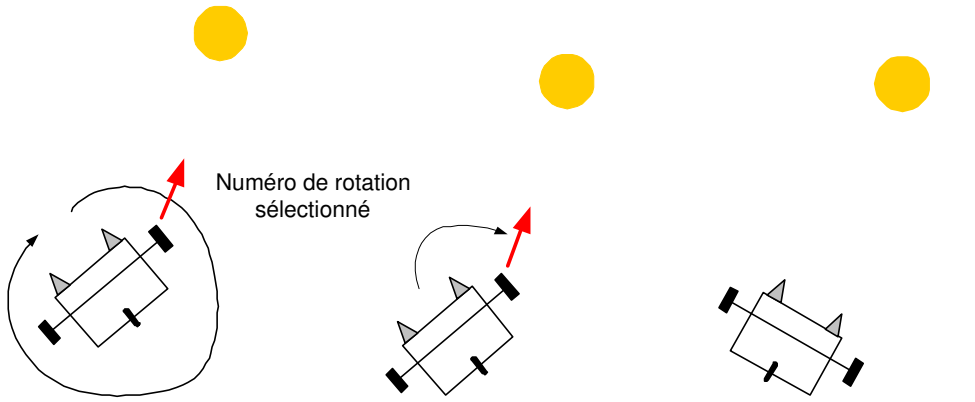


FIG. 5.8 – Première étape de la politique de contrôle figée. Elle est composée du tour complet et de la rotation de positionnement vers la source lumineuse.

Durant les étapes suivantes, une opération similaire est effectuée à la différence que le robot n'effectue pas un tour complet mais explore un ensemble restreint de directions en face de lui. En effet, une fois le tour complet effectué, nous supposons que le robot se trouve sur l'axe vers la source lumineuse avec une erreur n'excédant pas un certain angle. Cette hypothèse est vérifiée par l'ajustement de la distance parcourue lors de la phase de rapprochement expliquée par la figure 5.6.

Une fois le robot convenablement positionné, une action de déplacement permet au robot d'avancer vers la source lumineuse. Le robot s'arrête lorsque le nombre d'itérations effectuées atteint l'horizon c'est-à-dire le nombre maximal d'itérations.

Le pseudocode de l'algorithme est donné par l'algorithme 1. Un pseudocode plus détaillé est donné en annexe J.1.1.

Nous pouvons d'ores et déjà noter que de nombreuses opérations, à priori non nécessaires, doivent être effectuées pour mener à bien la tâche de contrôle. De plus, l'analyse du senseur de luminosité LEGO Mindstorms en section 4.6 a montré la variance importante des mesures bruitées. L'efficacité de cet algorithme, décrite lors du chapitre 6 traitant des expériences, est tributaire de

Algorithme 1 Pseudocode de l'algorithme pour la politique de contrôle figée.

```
FOR étape = 1 à horizon
  //Phase de positionnement
  IF étape = 1 THEN
    FOR toutes les rotations du tour complet
      Prendre les deux mesures de luminosité l1 et l2
      Sauvegarder le numéro de la rotation telle que l1+l2 soit minimum
    ENDFOR
  ELSE
    FOR toutes les rotations de l'ensemble restreint de rotations devant le robot
      Prendre les deux mesures de luminosité l1 et l2
      Sauvegarder le numéro de la rotation telle que l1+l2 soit minimum
    ENDFOR
  ENDIF
  //Phase de rapprochement
  Rapprocher le robot dans la direction spécifiée par le numéro de rotation sauvegardé
ENDFOR
```

la précision de ces mesures. Nous verrons dans le chapitre 6 traitant des expériences qu'il en résulte un comportement parfois erratique du robot nuisant considérablement aux performances.

5.3 Politique de Contrôle Basée sur l'Analyse des Données Sensorielles

Contrairement à la politique de contrôle figée, une analyse détaillée des données sensorielles permettra une meilleure compréhension du phénomène. Nous nous pencherons principalement sur l'amélioration de la recherche de la direction vers la source lumineuse en utilisant des outils statistiques. L'utilisation d'outils statistiques est motivée par la variance élevée des mesures bruitées de luminosité (voir section 4.6). Bien que l'introduction du concept de senseur virtuel (voir section 4.10) permette d'améliorer la précision des mesures de luminosité, la variance reste élevée et doit être prise en considération.

La figure 5.9 représente la politique de contrôle basée sur l'analyse des données sensorielles.

5.3.1 Analyse des Rotations Successives

Une collecte de données est réalisée afin de mettre en évidence les caractéristiques utiles du phénomène étudié. Le robot est placé à un endroit quelconque. Une série de 300 rotations sont effectuées et chaque rotation fait l'objet d'une mesure de luminosité par les deux senseurs.

La figure 5.10 représente les mesures à chaque rotation. Nous observons la répétition d'un motif correspondant à chaque nouveau tour complet du robot.

Le motif est subdivisé en trois zones auxquelles correspondent une série de positions du robot. Les figures 5.11 et 5.12 montrent la correspondance entre les positions du robot et les mesures lumineuses à chaque rotation.

La répétition du motif est indépendante de la position du robot contrairement à sa forme. En effet, la figure 5.13 représente une série de mesures prises d'une autre position. Bien que nous observions toujours la périodicité du motif, son profil est différent. Il en sera tenu compte dans l'algorithme de contrôle.

Un agrandissement sur un motif de la figure 5.10 ainsi que la superposition des mesures des deux senseurs sont représentés par la figure 5.15 : le motif des mesures des deux senseurs, bien que similaire, subit un décalage entre les deux mesures de luminosité des senseurs virtuels gauche et droit. Ces mesures de luminosité sont notées respectivement x_1 et x_2. Dans le cas contraire, l'intérêt de disposer de deux senseurs serait nul car leur comportement serait identique. La figure 5.14

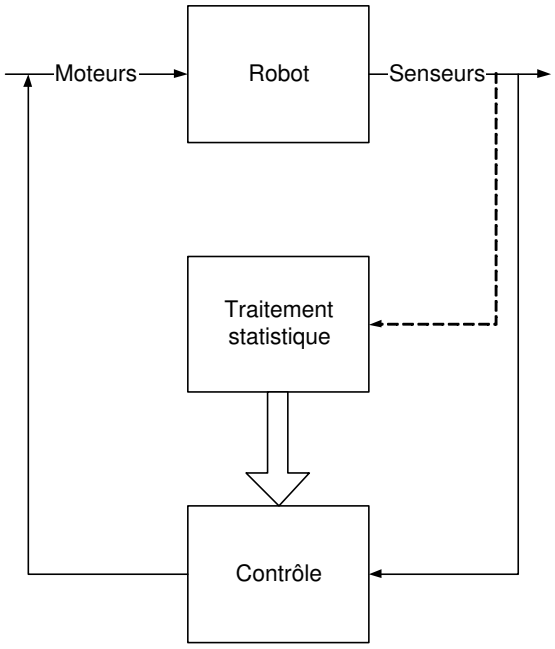


FIG. 5.9 – Politique de contrôle basée sur l'analyse des données sensorielles. La flèche en pointillé représente la collecte de données sensorielles.

montre deux positions possibles des senseurs de luminosité à l'avant du robot. La première position n'est pas intéressante car les mesures de luminosité des senseurs gauche et droit sont similaires. Au contraire, la deuxième position permet d'obtenir deux mesures de luminosité significativement différentes et ainsi, caractérise mieux la position du robot. Pour des mesures prises à chaque rotation d'amplitude unitaire, il existe un décalage d'environ huit rotations qui dépend de la zone dans laquelle se trouve le robot. Si le robot est dans la direction opposée à la source lumineuse, aucune information lumineuse, et par conséquent aucun décalage n'est enregistré.

Ce décalage peut aussi être mis en évidence en calculant la différence entre x_1 et x_2. La figure 5.16 montre un motif périodique qui découle directement des figures 5.15 et 5.10.

5.3.2 Analyse des Données Sensorielles

Les précédentes données collectées ont mis en évidence une relation entre les rotations et les mesures de luminosité. Son exploitation dans le cadre de la recherche de la direction vers la source lumineuse se base sur deux concepts distincts :

1. La différence entre x_1 et x_2 est nulle lorsque le robot se trouve en face de la source lumineuse. La direction de la source lumineuse est caractérisée par cette différence nulle. Ce concept est à la base de la technique de la différence nulle.
2. La différence entre x_1 et x_2 est estimée et utilisée en tant que telle. Ce concept est à la base de la technique de l'estimation de la différence.

5.3.2.1 Technique de la Différence Nulle

Idéalement, il existe une ou plusieurs rotations en face de la source lumineuse pour lesquelles la différence entre x_1 et x_2 est nulle. Il en est autrement dans le cas réel car l'amplitude des rotations, bien que faible, ne permet pas toujours de rencontrer ce cas de figure. Nous pouvons maintenant définir le problème à résoudre.

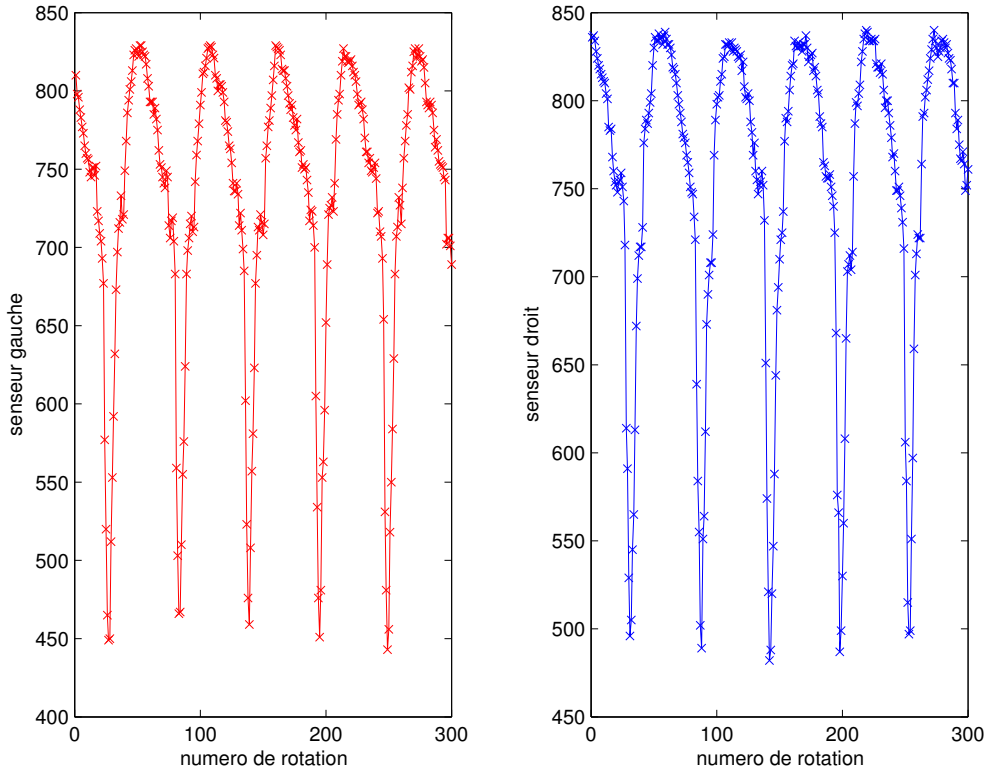


FIG. 5.10 – Mesures prises lors de 300 rotations séquentielles d'amplitude unitaire. x_1 et x_2 sont traités indépendamment. L'axe des abscisses représente la séquence des rotations notées par leur numéro. L'axe des ordonnées représente la mesure de luminosité par le senseur virtuel.

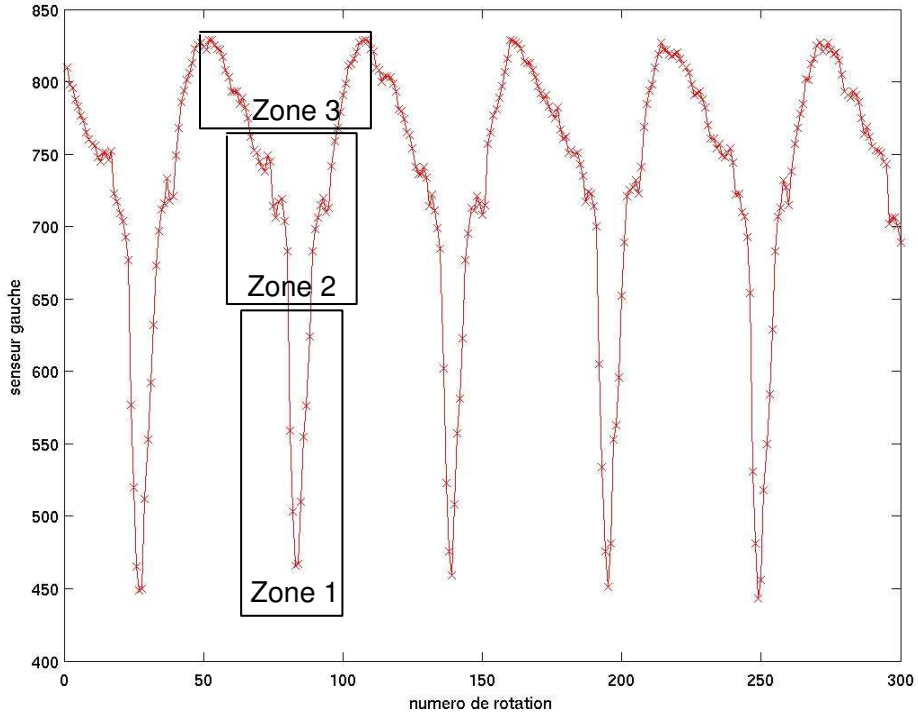


FIG. 5.11 – La succession des mesures prises lors des 300 rotations séquentielles d'amplitude unitaire, est subdivisée en trois zones. L'axe des abscisses représente la séquence des rotations notées par leur numéro. L'axe des ordonnées représente la mesure de luminosité par le senseur virtuel.

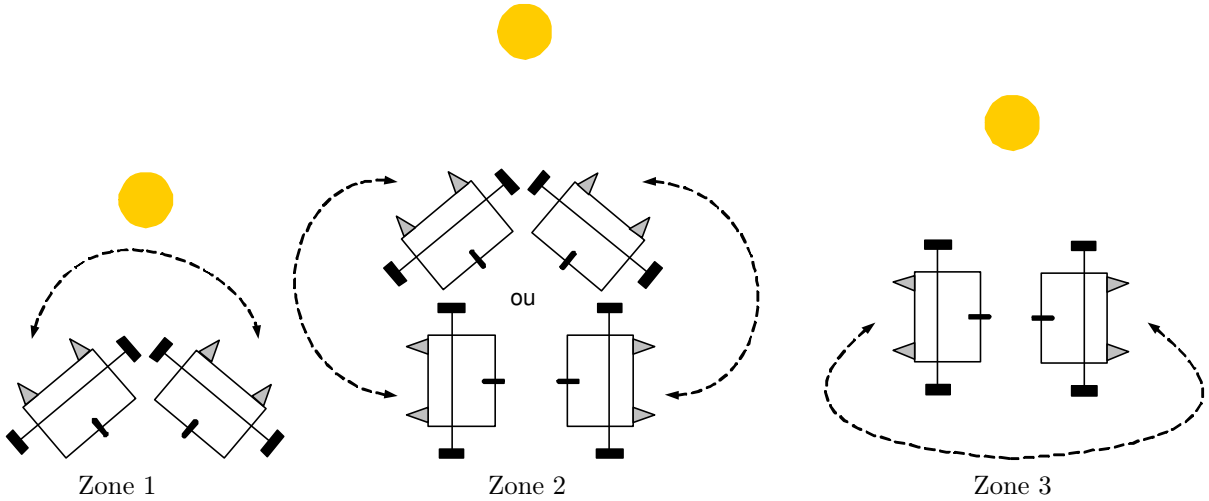


FIG. 5.12 – Positions correspondantes du robot dans les trois zones de la figure 5.11.

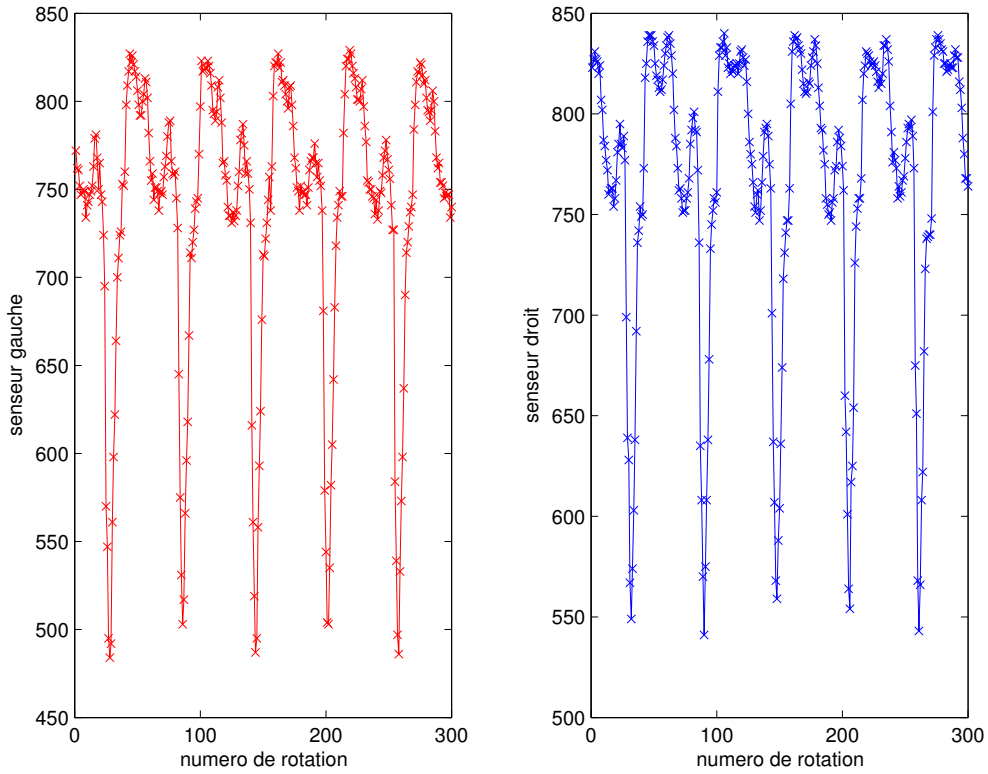
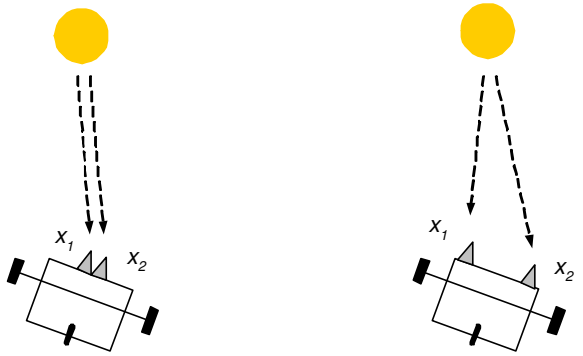


FIG. 5.13 – Mesures prises lors de 300 rotations séquentielles d'amplitude unitaire. x_1 et x_2 sont traités indépendamment. Une position différente de la figure 5.10 est considérée. L'axe des abscisses représente la séquence des rotations notées par leur numéro. L'axe des ordonnées représente la mesure de luminosité par le senseur virtuel.



La différence de luminosité $x_1 - x_2$ n'est pas significative.

La différence de luminosité $x_1 - x_2$ est significative.

FIG. 5.14 – Deux positions possibles des senseurs et leurs différences de luminosité résultantes.

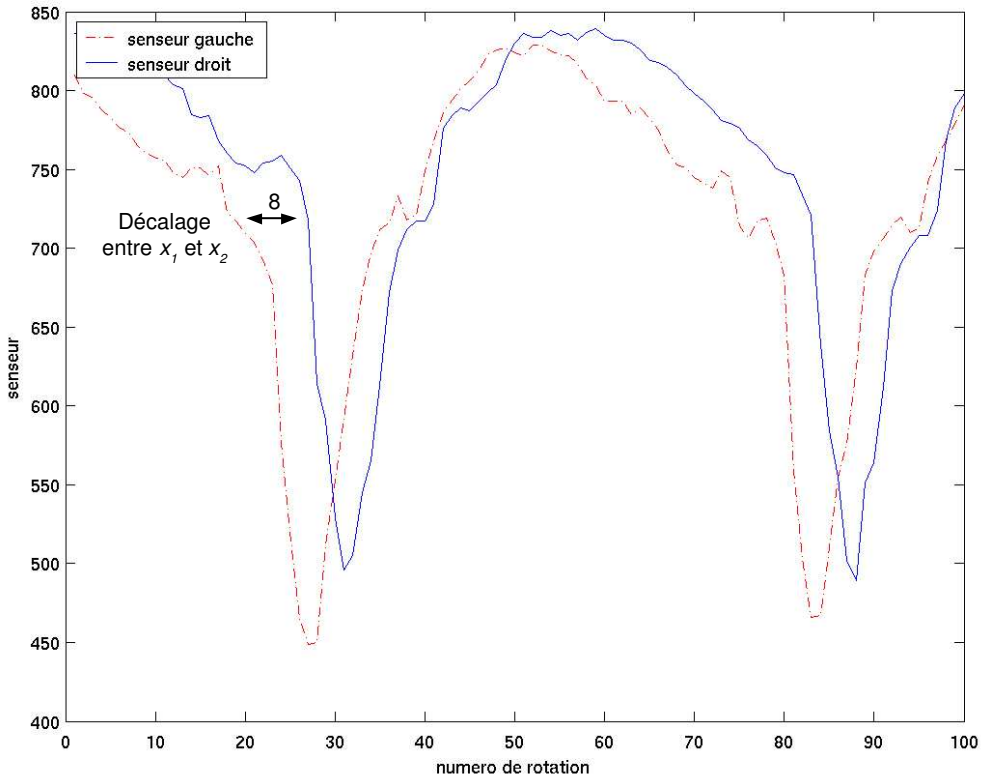


FIG. 5.15 – Agrandissement sur les mesures prises lors de 300 rotations séquentielles d'amplitude unitaire. x_1 et x_2 sont tracés sur la même figure afin d'observer leur décalage. L'axe des abscisses représente la séquence des rotations notées par leur numéro. L'axe des ordonnées représente la mesure de luminosité par le senseur virtuel.

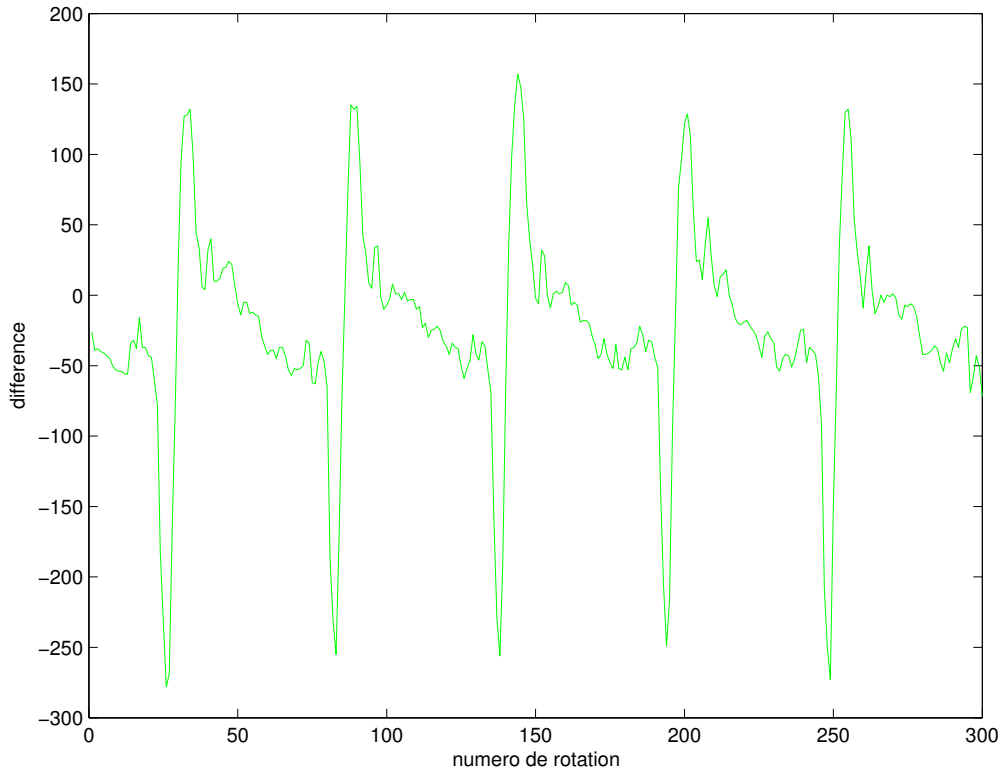


FIG. 5.16 – Différence entre les mesures des 300 rotations d'amplitude unitaire. L'axe des abscisses représente la séquence des rotations notées par leur numéro. L'axe des ordonnées représente la différence entre les mesures de luminosité des deux senseurs virtuels.

Problème : Tester s'il existe une différence nulle entre x_1 et x_2, correspondant à une rotation donnée.

Méthode : La variance des mesures bruitées ne permet pas une simple comparaison entre x_1 et x_2. Il convient donc de réaliser un test statistique [84, 43] qui permettra de déterminer avec un certain niveau significatif, le bien-fondé de l'hypothèse de la différence nulle entre x_1 et x_2.

La détermination du type de test statistique dépend de la nature de la distribution à traiter. A cette fin, une nouvelle collecte de données est réalisée :

- Un nombre arbitraire[1] de rotations sont effectuées.
- L'amplitude des rotations est unitaire. Etant donné que cette amplitude est minimale, les positions en face de la source lumineuse seront plus nombreuses.
- A chaque rotation, un nombre arbitraire[2] de mesures de luminosité sont effectuées.

Une estimation de la distribution de probabilité des mesures à chaque étape nous renseigne sur le type de distribution. La figure 5.17 donne cette estimation ainsi que l'histogramme des mesures pour la rotation numéro 28. D'autres estimations sont données en annexe E.

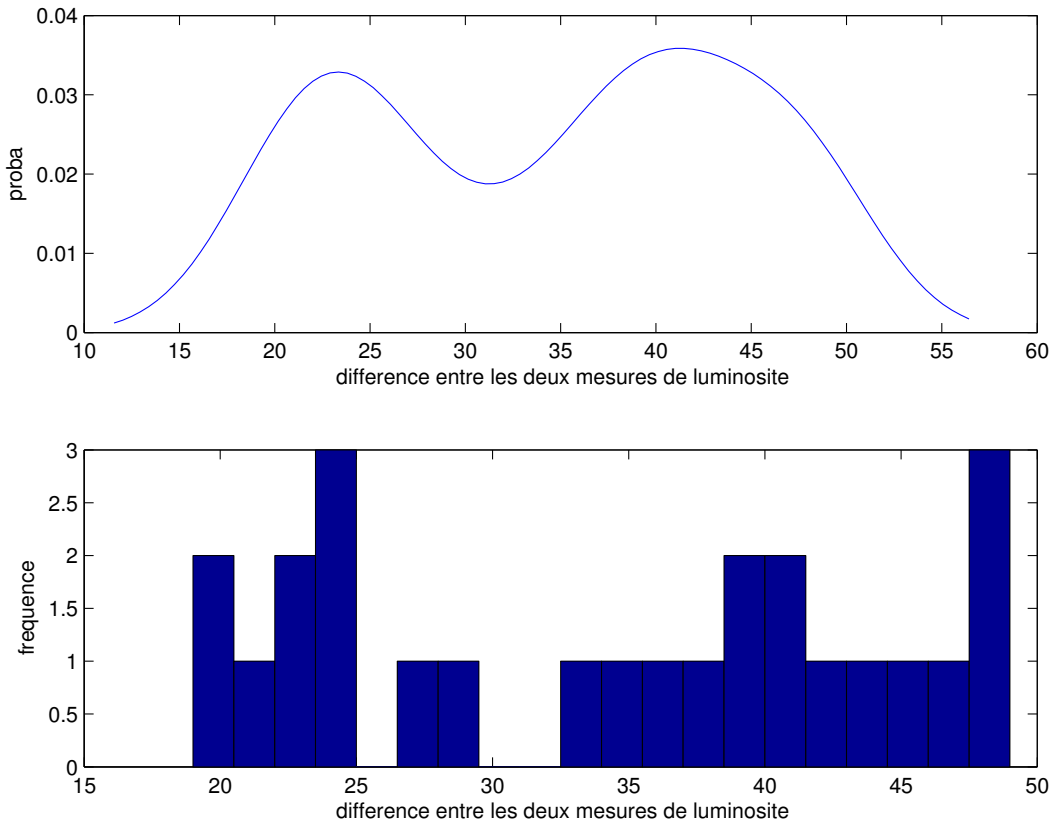


FIG. 5.17 – Estimation de la distribution de probabilité et histogramme des mesures de luminosité pour la rotation numéro 28.

Aucune d'entre elles ne révèle de distributions connues ce qui nous suggère l'utilisation de méthodes statistiques non-paramétriques [17, 84]. En effet, aucune présomption quant à la forme de la distribution de probabilité des mesures ne peut être formulée. En outre, la quantité de données est limitée ce qui rend les statistiques paramétriques peu fiables [17, 84].

[1] 55 rotations permettent au robot d'effectuer un tour complet. Le passage devant la lumière est donc certain.
[2] 25 mesures sont un compromis entre durée de l'expérience et quantité d'informations nécessaires à l'étude statistique.

Parmi les tests statistiques disponibles, les tests de permutation [43] sont adéquats à la résolution du problème posé.

Test de Permutation sur les Distributions des Senseurs

Afin de tester la présence d'une ou plusieurs positions dont la différence entre x_1 et x_2 est nulle, des tests de permutation [43, 39] sont calculés à partir des deux distributions x_1 et x_2. Ces tests permettent de vérifier l'hypothèse nulle que les deux distributions sont identiques. L'applicabilité des tests de permutation repose sur la propriété que certaines observations sont échangeables sous l'hypothèse nulle ce qui n'est pas le cas sous l'hypothèse alternative. Un choix d'hypothèse et une procédure spécifique, fourni en annexe F, réalisent les tests de permutation.

Faisons à présent la distinction entre la zone lumineuse et la zone obscure. Cette classification des mesures permettra la distinction entre les mesures contenant une information de lumière exploitable et les mesures qui se font dans l'obscurité. Une limite de luminosité *lim_ lum* est fixée[3] telle que :

- Si la mesure de luminosité est plus petite ou égale à *lim_ lum*, la zone est lumineuse.
- Sinon, la zone est obscure.

Résultats : Malheureusement, sur plusieurs ensembles de données collectées, seuls les tests de permutation dans la zone obscure sont vérifiés. En effet, les mesures dans la zone obscure sont souvent identiques car aucune information de lumière n'y est mesurée. Les résultats détaillés ainsi que le code source du script MATLAB sont fournis en annexe F.2.

Utilisation : Cette technique aurait pu être utilisée par un algorithme de contrôle. La direction de la source lumineuse serait alors déterminée par des tests de permutation. Les résultats ont montré qu'aucune rotation ne passait le test de permutation dans la zone lumineuse ce qui rend cette technique inutilisable par un algorithme de contrôle. Cependant, un changement de statistique rend possible son utilisation en introduisant de nombreuses contraintes. Cette variante est présentée en annexe F.3.

5.3.2.2 Technique de l'Estimation de la Différence

Nous allons à présent étudier la valeur quantitative des différences entre x_1 et x_2. Les tests de permutation précédents ont montré que la probabilité que les différences entre x_1 et x_2 soient nulles, est faible. Définissons le problème à résoudre.

Problème : Estimer la différence entre x_1 et x_2 par l'estimation d'une statistique arbitraire comme la moyenne empirique des différences. Cette estimation fournit une information quantitative exploitable par un algorithme de contrôle.

Méthode : L'estimation est réalisée par une méthode statistique non-paramétrique appelée procédure de rééchantillonnage de type *bootstrap* [39]. Le choix d'une technique de rééchantillonnage est motivée par le mêmes critères que pour les tests de permutation :

- la variance des mesures de luminosité bruitées est importante. L'utilisation de méthodes statistiques est donc indispensable.
- La distribution des mesures de luminosité est inconnue. Aucune hypothèse la concernant ne peut être posée (voir figure 5.17).
- Le nombre de mesures de luminosité dans chaque échantillon est limité.

[3] Cette limite, déterminée par expérience, est fixée à 650.

Estimateur de la Différence entre les Senseurs

Le rééchantillonnage fut introduit par Efron [8] et permet de juger de la qualité d'un estimateur notamment par le calcul du biais, de la variance et de l'intervalle de confidence. La technique de *bootstrap* se base sur un rééchantillonnage des données originales avec remplacement. Le détail ainsi que le code source du script MATLAB sont donnés en annexe G.

Résultats : La table G.1, donnée en annexe G.2, est un exemple de résultat obtenu par application de la technique *bootstrap* sur les données observées.

Nous pouvons déduire des résultats la bonne précision de l'estimateur de la moyenne. L'utilisation de cet estimateur devrait améliorer la performance de l'algorithme de contrôle en fournissant une meilleure estimation de la différence de la luminosité.

Utilisation : Il s'agit donc de rechercher la différence de luminosité la plus proche de zéro. En étudiant les estimateurs calculés, nous pouvons nous limiter à la zone lumineuse (voir section 5.3.2.1).

La figure 5.18 représente l'évolution de l'estimateur lors d'une rotation complète.

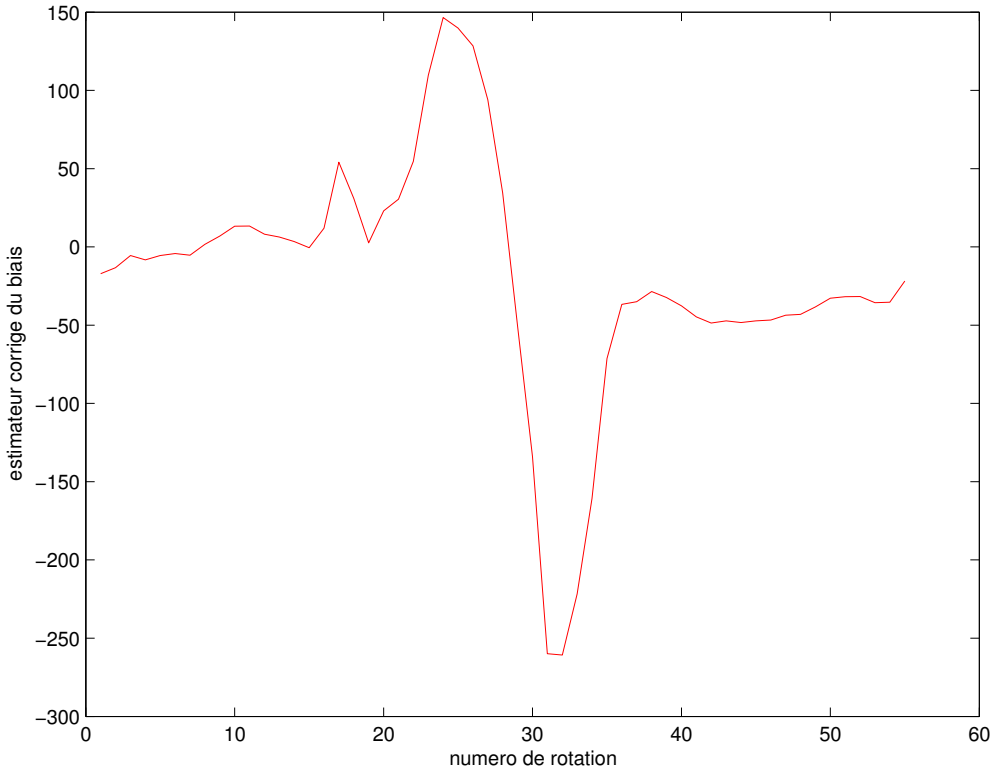


FIG. 5.18 – Evolution de l'estimateur de la moyenne de la différence de luminosité durant une rotation complète.

Deux solutions s'offrent à nous :

1. Considérons la valeur absolue de l'estimateur (voir figure 5.19). La recherche de la position en face de la lumière se résume à la localisation du minimum global. Cependant, il se pose le problème de la présence de minima locaux ce qui complique considérablement la recherche. Appelons cette opération, la technique de recherche du minimum, le minimum global étant la position en face de la source lumineuse.

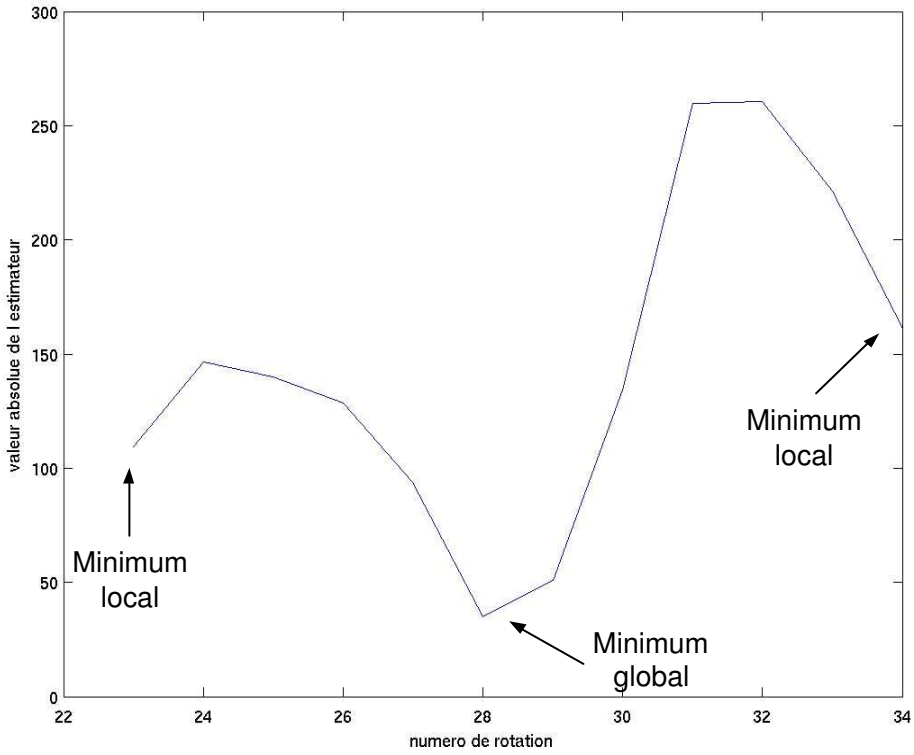


FIG. 5.19 – Valeur absolue de l'estimateur de la moyenne de la différence de luminosité dans la zone lumineuse. Dans cet ensemble de données, la zone lumineuse se résume aux rotations 23 à 34.

2. Recherche du passage par zéro. En effet, dans la zone lumineuse, seuls les estimateurs de deux rotations effectuent la transition du positif au négatif, passant ainsi par zéro. La position recherchée est donc une de ces deux rotations dont l'estimateur est le plus petit en valeur absolue. Cette technique évite le problème des minima locaux. Sa mise en oeuvre est cependant plus lourde comme nous le verrons dans le pseudocode de l'algorithme 2. Appelons cette opération, la technique de recherche du passage par zéro.

Ces deux techniques sont implémentées dans l'algorithme de contrôle. Leurs rôles sont décrits dans la section ci-dessous.

5.3.3 Algorithme de Contrôle

Le déroulement de l'algorithme de contrôle est similaire à celui de la politique de contrôle figée. La recherche par rotations successives est différente sur plusieurs points :

- Plusieurs mesures de luminosité sont réalisées à chaque rotation. La technique de *bootstrap* permet le calcul de l'estimateur décrit précédemment.
- La recherche ne se fait plus par la sélection d'un minimum parmi plusieurs mesures mais par les techniques de recherche du minimum et du passage par zéro.

Lors de la première étape, le robot doit se trouver dans une zone obscure. Si ce n'est pas le cas, une ou plusieurs rotations seront effectuées pour l'y amener. Une fois le robot dans la zone obscure, une succession de rotations permettront de détecter le passage par zéro et de sélectionner la position optimale.

Un premier déplacement rapprochera le robot de la source lumineuse. Après ce déplacement, la direction du robot est proche de la position optimale ce qui évitera de traiter le cas des minima locaux. Il suffit donc d'effectuer une recherche du minimum global.

Le pseudocode de l'algorithme est donné par l'algorithme 2. Un pseudocode plus détaillé est donné en annexe J.2.

Algorithme 2 Pseudocode de l'algorithme pour la politique de contrôle basée sur l'analyse des données sensorielles utilisant la technique de *bootstrap* afin d'estimer la moyenne de la différence de luminosité. La variable *horizon* est fixée à 50, *nbr_ mesures* à 25, les actions a et b sont fixées respectivement à 2 et 5.

```
FOR étape = 1 à horizon
  //Phase de positionnement
  IF étape = 1 THEN
    IF robot dans la zone obscure THEN
      Effectuer la recherche du passage par zéro
    ELSE
      Amener le robot dans la zone obscure
    ENDIF
  ELSE
    Recherche du minimum global de la valeur absolue de l'estimateur
  ENDIF
  //Phase de rapprochement
  Rapprocher le robot dans la direction choisie
ENDFOR
```

Les performances obtenues avec cet algorithme seront décrites lors du chapitre 6 traitant des expériences. Bien que la recherche de la direction de la source lumineuse utilise moins de rotations que la politique de contrôle figée, de nombreuses opérations non nécessaires sont encore effectuées.

5.4 Politique de Contrôle Basée sur l'Apprentissage de la Relation Sensori-Motrice

Contrairement aux politiques de contrôle précédentes, la politique d'apprentissage du contrôle ne se base sur aucune connaissance antérieure tant au niveau de la cinématique du robot que des senseurs. Nous nous baserons cette fois sur des données collectées afin de construire un modèle de la relation sensori-motrice. Il s'agit donc d'une politique de contrôle indirect [6] car toute sélection d'action s'effectue après consultation du modèle. Un algorithme de contrôle utilisant ce modèle est mis en oeuvre et ses performances seront décrites au chapitre 6 traitant des expériences. La figure 5.20 représente la politique de contrôle basée sur l'apprentissage de la relation sensori-motrice.

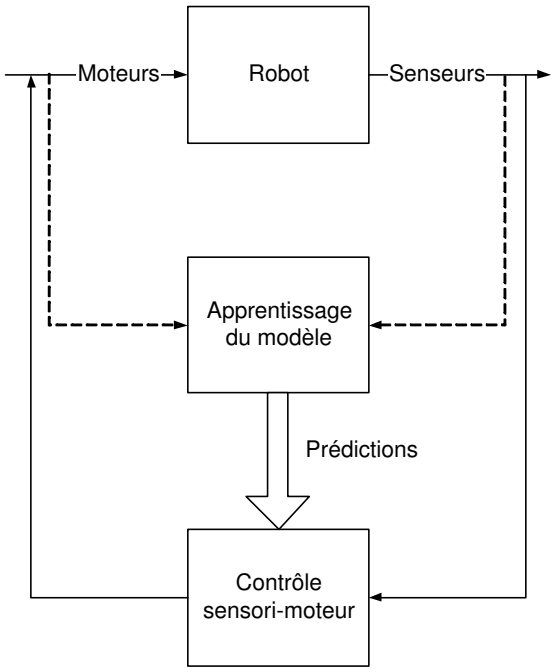


FIG. 5.20 – Politique de contrôle basée sur l'apprentissage de la relation sensori-motrice. La flèche en pointillé représente la collecte de données sensorielles.

Comme le montre la figure 5.21, le processus de modélisation est subdivisé en deux phases distinctes : la phase préliminaire et la phase d'apprentissage.

Détaillons la phase préliminaire regroupant la formulation du problème étudié, le plan expérimental et le prétraitement.

Formulation du problème. Nous formalisons la tâche de recherche de la source lumineuse comme un problème de contrôle d'un système dynamique. Nous considérons la relation sensori-motrice comme un système dynamique à temps discret de la forme

$$x(k+1) = f_{k+1}\left(x(k),\, x(k-1),\, x(k-2),\, ...,\, x\left(k-\delta_x\right),\, u(k),\, u(k-1),\, u(k-2),\, ...,\, u\left(k-\delta_u\right)\right)+w(k) \tag{5.1}$$

$$u(k) = g_k\left(x(k),\, x(k-1),\, x(k-2),\, ...,\, x\left(k-\delta_x\right)\right) \tag{5.2}$$

tel que :

- $k = 0, 1, ..., N-1$ est un instant du temps discret. La discrétisation du temps est une hypothèse simplificatrice du phénomène. En effet, l'environnement du robot étant le monde réel, le temps est continu.

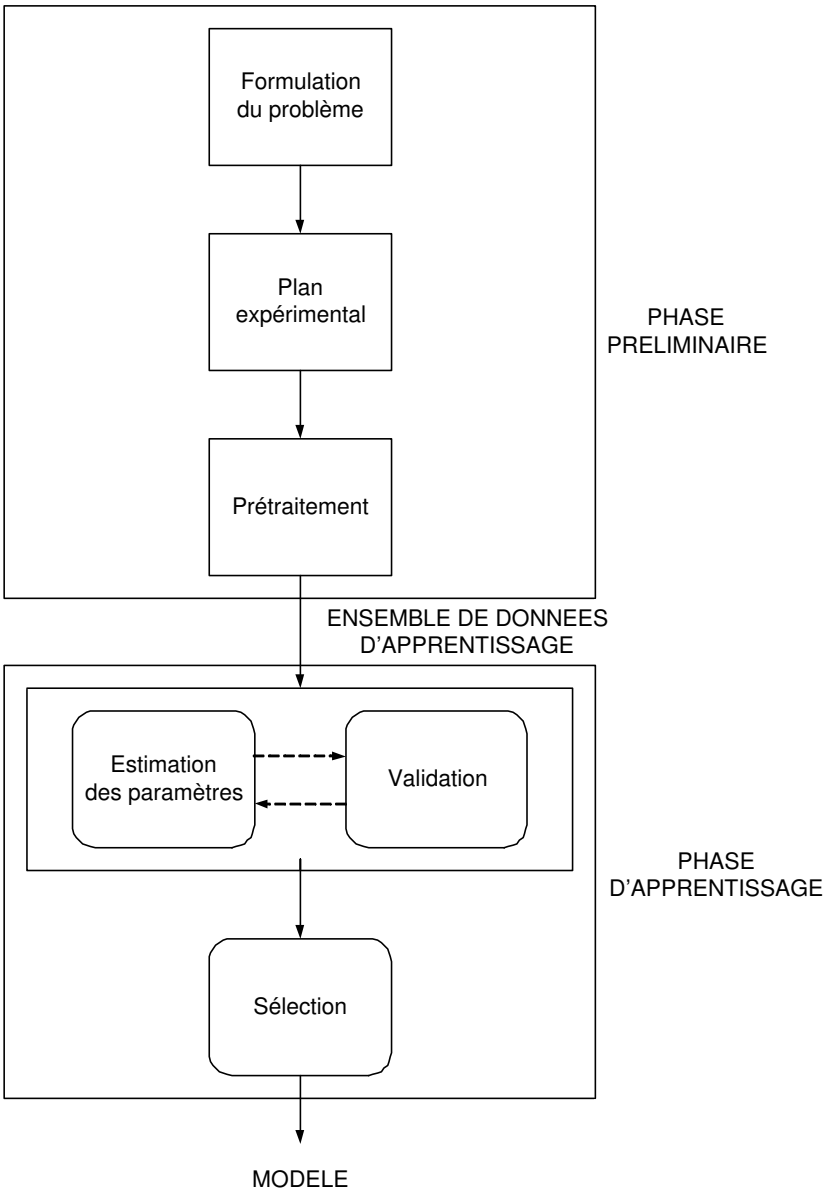


FIG. 5.21 – Processus d'apprentissage d'un modèle à partir des données expérimentales et sa décomposition en phase préliminaire et phase d'apprentissage.

- N est l'horizon, c'est-à-dire le nombre de fois qu'une action est appliquée. L'horizon permet donc de limiter le nombre d'actions effectuées par le contrôle du robot et ainsi y mettre fin.
- δ_x et δ_u sont les décalages temporels respectivement pour l'état x du système et pour l'action de contrôle u.
- $f_k(.,.,...,.)$ est une fonction quelconque. Nous verrons plus tard que le choix du type de fonction f déterminera le choix du modèle sous-jacent les données. Notons que si f est linéaire, nous utiliserons un modèle ARX sinon $NARX$.
- $x(k) \in S_k$ est l'état du système. S_k est l'ensemble des états possibles à l'instant k. Les états du système aux instants k, $k-1$, $k-2$, ..., $k-\delta$ résument toutes les informations nécessaires et suffisantes pour décrire l'évolution du système. Pour la tâche de contrôle considérée, nous supposons δ_x et δ_u nuls. Les mesures de luminosité à l'instant k contiennent donc toute l'information pertinente afin de modéliser le comportement futur du système. Notons que les spécificités techniques des senseurs de luminosité LEGO Mindstorms (voir section 4.6) limitent la taille de l'ensemble des états du système aux couples $[x_1, x_2]$ tel que $x_1 \in [0, 1024]$ et $x_2 \in [0, 1024]$.
- $u(k) \in U\,(x(k))$ est l'action de contrôle appartenant à l'ensemble des actions de contrôle possibles U selon l'état du système. Notons qu'il existe une contrainte sur les actions de contrôle : les choix effectués lors de la mise au point du contrôle des moteurs (voir section 4.2) limitent l'ensemble des actions aux couples $[u_1, u_2]$ tel que $u_1 \in [-255, 255]$ et $u_2 \in [-255, 255]$. De plus, l'algorithme de contrôle ne considère que les actions de rotation (voir section 5.1) impliquant que $u_1 = -u_2$.
- g_k est la fonction de contrôle calculant l'action à effectuer selon l'état du système.
- $w(k)$ représente le bruit à l'étape k. Nous supposons que les valeurs $w(k)$ sont équi-distribuées et indépendantes, que $E\,[w] = 0$ et que $Var\,[w]$ est finie.

Le système fonctionne en boucle fermée (voir figure 5.20). Contrairement aux systèmes fonctionnant en boucle ouverte où la séquence des actions de contrôle est entièrement déterminée à l'instant initial, le système étudié permet de considérer l'état courant afin d'ajuster les actions.

Le choix de l'état du système comme étant le couple des mesures de luminosité des senseurs gauche et droit, est arbitraire. Nous aurions pu considérer la position du robot comme étant l'état caché du système obtenue grâce à des techniques telles que le filtrage de Kalman [88, 57]. Cependant, les informations sensorielles sont très limitées ce qui nous suggère de réduire l'état du système au couple de mesures choisi.

La relation sensori-motrice est formulée comme suit :

Soit $x_1(k)$ et $x_2(k)$ les mesures de luminosité à l'instant k respectivement sur les senseurs de luminosité gauche et droit, notés l_1 et l_2.

Soit $u_1(k)$ et $u_2(k)$ les actions effectuées à l'instant k respectivement sur les moteurs gauche et droit, notés m_1 et m_2.

La fonction f_{k+1} dans l'équation 5.1 peut être décomposée en deux relations, chacune modélisant le système pour un senseur :

$$\begin{cases} x_1(k+1) = p_{k+1}\,(x_1(k),\, x_2(k),\, u_1(k),\, u_2(k)) + w_1 \\ x_2(k+1) = q_{k+1}\,(x_1(k),\, x_2(k),\, u_1(k),\, u_2(k)) + w_2 \end{cases} \tag{5.3}$$

Ces deux relations seront traitées de façon identique. Pour la simplicité de l'exposé, nous nous focaliserons sur la relation

$$x(k+1) = h_{k+1}\,(x_1(k),\, x_2(k),\, u_1(k),\, u_2(k)) + w \tag{5.4}$$

où x représente indifféremment x_1 ou x_2.

Plan expérimental. Le plan expérimental est la procédure qui permet d'obtenir un ensemble de données sensées être les plus représentatives du phénomène afin d'optimiser le processus de modélisation. Il s'agit dans ce cas-ci des différentes techniques d'acquisition de données par l'intermédiaire du robot. Trois techniques d'acquisition de données sont mises au point :

1. Acquisition aléatoire.
2. Acquisition assistée.
3. Acquisition par rotation.

Elles nous permettent d'obtenir un ensemble de données d'apprentissage représentatives du phénomène étudié. Elles sont décrites en détail au chapitre 6 traitant des expériences.

Prétraitement. Le prétraitement des données brutes est effectué par les senseurs virtuels abordée en section 4.1.2.4 qui permettent de minimiser la sensibilité des senseurs au bruit et d'éviter les mesures hors-normes.

Une fois l'ensemble des données d'apprentissage collecté à la fin de la phase préliminaire, la phase d'apprentissage commence. Celle-ci est composée essentiellement d'une recherche de modèle représentant au mieux le phénomène sous-jacent les données. A des fins comparatives, des critères de validation seront définis.

5.4.1 Modélisation

Il existe de nombreuses techniques de modélisation, la littérature étant très fournie sur ce sujet [7, 16, 26, 44]. Seuls deux modèles seront étudiés et leur comparaison permettra la sélection du plus performant. Ils ont été choisis pour leurs caractéristiques intrinsèques et leurs performances potentielles.

5.4.1.1 Modèle Linéaire

Le choix d'un modèle linéaire se base sur l'hypothèse que le système est linéaire, c'est-à-dire du type

$$x(k+1) = \beta_0 + \beta_1 x_1(k) + \beta_2 x_2(k) + \beta_3 u_1(k) + \beta_4 u_2(k) + w \tag{5.5}$$

La régression linéaire [36] est la technique statistique la plus classique et la plus répandue pour modéliser une relation linéaire. Dans le cas de la relation 5.4, la présence de plusieurs variables d'entrée mène à l'utilisation d'une régression linéaire multiple.

Voici quelques avantages de la régression linéaire pour la modélisation, la prédiction ou une tâche de contrôle.

Méthode rapide. Plusieurs techniques telles que la technique des *moindres carrés* (*LS*) [30, 38], mettent en oeuvre la régression linéaire rapidement et efficacement. La construction du modèle et sa validation sont donc à la portée de la puissance de calcul actuelle et du problème considéré.

Apprentissage en ligne La technique récursive des moindres carrés, notée *RLS* [17], permet de mettre à jour efficacement le modèle lors de l'ajout de nouvelles données.

Peu de mémoire. Les données ne sont utilisées que lors de l'ajustement du modèle. Elles ne doivent donc pas être gardées.

Voici quelques désavantages de la régression linéaire.

Hypothèses. La régression linéaire ne donnera de bonnes performances que si l'hypothèse de linéarité du système est vérifiée. Dans le cas contraire, la faculté de prédiction de modèle sera mauvaise. De plus, la régression linéaire est effectuée sur l'ensemble des données. il est donc fait l'hypothèse de l'existence d'une fonction linéaire globale aux données.

Calculs. Toute la puissance de calcul est requise lors de l'ajustement du modèle. Une fois ce calcul réalisé, une faible puissance de calcul est suffisante. Ceci peut être désavantageux si la quantité de données est importante car la phase d'ajustement demandera beaucoup de puissance.

La technique de régression linéaire des moindres carrés, notée *LS* [30], permet l'estimation des paramètres β_i avec $i = 0, 1, 2, 3, 4$, de la relation 5.5 afin d'obtenir la relation

$$\hat{x}(k+1) = \hat{\beta}_0 + \hat{\beta}_1 x_1(k) + \hat{\beta}_2 x_2(k) + \hat{\beta}_3 u_1(k) + \hat{\beta}_4 u_2(k) \tag{5.6}$$

Connaissant l'état du système ($x_1(k)$ et $x_2(k)$) et l'action de contrôle ($[u_1(k), u_2(k)]$) à appliquer, nous pouvons prédire le résultat ($\hat{x}_1(k+1)$ et $\hat{x}_2(k+1)$) grâce à la relation 5.6.

La technique de validation de ce modèle sera abordée dans la section 5.4.1.3.

5.4.1.2 Modèle Non-Linéaire : Lazy Learning

Il existe de nombreuses méthodes non-linéaires de modélisation. On peut citer les réseaux de neurones non-linéaires [3], les régressions polynomiales de degré n, les splines ainsi que le Lazy Learning [16, 19, 20, 18]. Le choix s'est porté sur le Lazy Learning dont les avantages sont présentés ci-dessous. Nous renvoyons le lecteur aux références [16, 19, 20, 18] pour une description détaillée de la méthode.

Lazy Learning

Le Lazy Learning est une technique de modélisation basée sur la mémorisation des données (*memory-based*). L'ensemble des données observées doivent être gardées afin de pouvoir répondre aux requêtes. Le travail de modélisation est postposé à la requête c'est-à-dire que tout le calcul est effectué une fois la requête de prédiction reçue.

La prédiction est réalisée en interpolant localement les occurrences voisines de la requête qui sont considérées comme pertinentes. La pertinence des occurrences voisines est déterminée par une mesure de distance. Cette caractéristique implique que le Lazy Learning soit une technique de modélisation locale.

Voici les principaux avantages du Lazy Learning pour la modélisation, la prédiction ou une tâche de contrôle.

Peu d'hypothèses. Le Lazy Learning ne suppose aucune connaissance à priori du processus sous-jacent les données. La seule information disponible est représentée par un ensemble fini d'observations sous la forme d'entrée/sortie. Ceci est indispensable dans notre cas parce qu'aucune hypothèse ne peut être faite quant à la relation sensori-motrice.

Conception requête par requête Le problème de modélisation est résolu localement sans faire d'hypothèse sur l'existence d'une fonction globale décrivant les données et sans faire d'hypothèse sur la nature du bruit. A nouveau, nous ne pouvons faire l'hypothèse d'une fonction globale ou de la nature du bruit dans le cas étudié.

Le Lazy Learning utilise des méthodes linéaires dans des problèmes non-linéaires. Beaucoup de techniques et de résultats théoriques existent pour la statistique linéaire comme en témoigne la régression linéaire. Le Lazy Learning réutilise ses éléments dans une optique non-linéaire.

Apprentissage en ligne. Le Lazy Learning peut facilement gérer une tâche en ligne car il suffit d'ajouter les nouvelles données à l'ensemble de départ afin qu'elles soient prises en compte à la prochaine requête. Nous verrons que cette caractéristique s'avère très intéressante lors de la mise en oeuvre d'algorithmes de contrôle adaptatifs.

Le principal désavantage du Lazy Learning est le stockage de l'ensemble des données durant toute la durée des prédictions.

L'algorithme *LL* a été développé en MATLAB et en C par M. Birattari et G. Bontempi. Les détails d'utilisation du Lazy Learning ainsi que les fonctions en C ou les fonctions *MATLAB*, sont fournis dans le rapport technique [15]. Notons que nous utilisons la fonction du Lazy Learning la plus puissante à savoir la combinaison locale de modèles. Dans cette fonction, un modèle constant, linéaire et de second degré sont évalués et combinés afin de fournir un modèle donnant de bonnes prédictions.

5.4.1.3 Validation de Modèle

La validation de modèle vise à estimer l'erreur de généralisation du modèle G_N. En effet, un modèle fonctionne bien s'il montre une bonne faculté de généralisation. Il est donc capable d'effectuer de bonnes prédictions avec de nouvelles données ne faisant pas partie de l'ensemble des données d'apprentissage. L'estimation de G_N permet une sélection entre plusieurs modèles.

Le concepteur n'ayant pas accès à l'erreur de généralisation (il faudrait connaître la nature du bruit par exemple [17]), plusieurs techniques d'estimation peuvent être utilisées telles que les techniques de rééchantillonnage comme le *bootstrap* [8] et la validation croisée [83, 87].

Dans le cadre de ce mémoire, nous utiliserons un cas particulier de validation croisée appelée *leave-one-out* (*L-O-O*) [75]. L'idée de la validation croisée de type *L-O-O* est de construire un modèle à partir des données diminuées de l'une d'entre elles et d'utiliser ce modèle pour prédire la donnée retirée. L'ensemble des données original D_N est divisé N fois en un sous-ensemble d'apprentissage contenant $N-1$ données et un sous-ensemble de validation contenant la donnée restante. Pour chaque paire des N sous-ensembles d'apprentissage et de validation, la prédiction $\hat{y}_i^{-i}$ du modèle pour la $i^{\text{ème}}$ observation est calculée. L'estimateur par validation croisée *L-O-O,* $\hat{G}_{loo}$, de l'erreur de généralisation est donné par

$$\hat{G}_{loo} = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - \hat{y}_i^{-i} \right)^2 \tag{5.7}$$

Les résultats de l'estimation de l'erreur de généralisation est donnée au chapitre 6 traitant des expériences. Les deux modèles seront comparés afin de sélectionner le plus performant pour réaliser l'algorithme de contrôle.

5.4.2 Contrôle

Le choix des actions de contrôle est effectuée par la fonction de contrôle g_k (voir paragraphe 5.4, équation 5.2). Cette fonction se base sur le modèle utilisé et sur une fonction coût, notée C.

L'apprentissage du modèle de la relation sensori-motrice permet de prédire les mesures de luminosité résultante de l'application d'une action à partir d'un état du système. La fonction coût C dépend de l'action, des mesures lumineuses avant action et des mesures lumineuses prédites. Elle permettra de sélectionner l'action la plus performante. La fonction coût C est de la forme

$$C = c\left(x_1(k),\, x_2(k),\, u_1(k),\, u_2(k),\, \hat{x}_1(k+1),\, \hat{x}_2(k+1)\right) \tag{5.8}$$

où $\hat{x}_1(k+1)$ et $\hat{x}_2(k+1)$ sont les prédictions de l'état du système à l'instant $k+1$ respectivement pour les mesures de luminosité du senseur gauche et droit.

L'espace des actions U étant restreint, un test de chaque action grâce au modèle permet le calcul de la fonction coût. L'action dont le coût est minimal est alors sélectionnée.

Lors du chapitre 6 traitant des expériences, nous verrons que le choix de la fonction coût est déterminante pour les performances de l'algorithme de contrôle. L'action de coût minimal est ensuite sélectionnée.

5.4.3 Algorithme de Contrôle

Le squelette de l'algorithme pour la politique d'apprentissage du contrôle est similaire aux politiques précédentes excepté une différence importante : le robot n'effectue plus d'exploration afin de trouver une bonne direction pour la source lumineuse mais réalise une action unique basée sur le modèle utilisé. Le nombre de rotation est donc minimal.

L'acquisition des données a permis de garder sous forme de fichier toutes les données d'apprentissage indispensable à l'ajustement du modèle. Ce fichier est décomposé en données d'entrée et de sortie.

Intervient ensuite l'apprentissage du modèle qui dépend de celui-ci : si le modèle linéaire est utilisé, l'ajustement consistera en une régression linéaire multiple par une technique *LS*. S'il s'agit du Lazy Learning, seul le stockage des données est nécessaire étant donné que le calcul du modèle est postposé à la requête de prédiction.

Afin de déterminer l'état initial du système, une première mesure de luminosité par les deux senseurs est effectuée.

Chaque étape est composée d'une phase de positionnement et d'une phase de rapprochement. La phase de positionnement est constituée d'une seule rotation dont l'amplitude sera déterminée par la fonction de contrôle. Notons que le modèle de prédiction n'est utilisé par la fonction de contrôle que dans le cas où le robot se trouve dans la zone lumineuse. En effet, l'absence d'information lumineuse dans la zone obscure ne permet pas la construction d'un modèle prédictif utile à la tâche de contrôle envisagée. Une action par défaut est entreprise lorsque le robot est dans une zone obscure, soit qu'une mauvaise prédiction l'y a amené, soit que telle était sa position initiale.

Le pseudocode de l'algorithme est donné par l'algorithme 3. Un pseudocode plus détaillé est donné en annexe J.3.

Algorithme 3 Pseudocode de l'algorithme pour la politique d'apprentissage du contrôle. Le modèle est soit le modèle linéaire, soit le Lazy Learning.

```
Ajustement du modèle de prédiction à partir des données d'apprentissage
Prendre les deux mesures de luminosité pour initialiser l'état du système
FOR étape = 1 à horizon
  //Phase de positionnement
  IF mesures dans la zone lumineuse THEN
    Utilisation du modèle pour sélectionner l'action de coût minimal
  ELSE
    Effectuer action de positionnement par défaut
  ENDIF
  Effectuer action sélectionnée
  Prendre deux nouvelles mesures de luminosité
  IF le robot n'est pas bloqué dans son mouvement THEN
    Enregistrer les nouvelles données dans le fichier des données d'apprentissage
  ENDIF
  //Phase de rapprochement
  Rapprocher le robot dans la direction choisie
ENDFOR
```

5.4.4 Apprentissage Adaptatif du Contrôle

L'adaptabilité de l'apprentissage du contrôle s'avère intéressante lorsque :

- Les nouvelles données prises lors du contrôle peuvent venir enrichir le modèle. En effet, l'ensemble des données d'apprentissage peut être très limité voire inexistant.
- Le phénomène varie avec le temps. Il en résulte que les données anciennes corroborent moins le phénomène que les données plus récentes.

5.4.4.1 Modèle Linéaire

Lors de la disponibilité de nouvelles données, le modèle préalablement ajusté est mis à jour grâce à la technique *RLS* [17, 69]. Cette version récursive de la technique *LS* permet une mise à jour rapide et évite un ajustement sur l'ensemble des données. L'ensemble des données ne doit donc pas être gardé.

De plus, le *RLS* fait intervenir un facteur d'oubli *f*. Si f est différent de l'unité[4], le poids des données lors de l'ajustement sera plus important pour les données les plus récentes. Cette propriété s'avère intéressante si le phénomène est non-stationnaire et linéaire ou s'il est stationnaire, non-linéaire mais peut être approximé par un modèle linéaire localisé dans le temps.

5.4.4.2 Modèle Non-Linéaire : Lazy Learning

L'enrichissement du modèle s'effectue par l'ajout des nouvelles données dans l'ensemble d'apprentissage des données d'entrée et de sortie.

L'adaptation au phénomène non-stationnaire est réalisée par un changement de la notion de voisinage dans le Lazy Learning. En effet, le voisinage ne se base plus sur une notion spatiale mais spatio-temporelle ce qui permettra une pondération des différents voisins selon leur distance et leur âge [16].

[4]La mise à jour simple du modèle linéaire par *RLS* est simplement un cas particulier de *RLS* avec f valant 1.

Chapitre 6

Expériences

Ce chapitre introduit les expériences réalisées avec le robot mobile. Cela concerne à la fois le Printemps des Sciences [32] et les expériences de contrôle et de communication réalisées au laboratoire dédié aux LEGO Mindstorms du Département Informatique de la Faculté des Sciences.

6.1 Environnement Expérimental

L'environnement expérimental est déterminant dans le cadre de la robotique mobile. Le matériel utilisé étant limité, l'adaptation de l'environnement permet d'améliorer la précision des éléments du robot tels que les senseurs, les moteurs et la communication infra-rouge. En effet, l'environnement du robot est le monde réel qui est :

- Inaccessible. Les senseurs sont imparfaits, leur portée limitée.
- Non-déterministe du point de vue du robot. Les roues dérapent, les batteries se déchargent, la surface n'est pas lisse ce qui impliquent qu'une action peut ne pas fonctionner.
- Non-stationnaire, l'effet d'une action change avec le temps.
- Dynamique. L'action doit-elle être reportée ou est-il plus performant d'agir immédiatement ?
- Continu. Dès lors, le nombre de configurations possibles est infini.

Nous nous focaliserons principalement sur la précision des mouvements, des mesures de luminosité et de la communication.

6.1.1 Laboratoire du Département d'Informatique

Ce laboratoire sera le théâtre des expériences les plus importantes à savoir le contrôle et la communication.

Les mouvements. Les pièces LEGO ne contiennent pas de petite roue mobile adéquate comme nous l'avons vu lors de la section 4.1.1 traitant de la cinématique du robot. La petite roue rigide utilisée est donc très sensible au type de surface. Il s'agit ici de limiter les frottements en choisissant une surface dure et lisse telle qu'une table. Plusieurs tables mises côte à côte constitue la surface accueillant le robot mobile. Mis à part les jointures entre les tables qui pose problème lors du passage du robot, cette surface est globalement satisfaisante.

Les mesures de luminosité. Le but des mesures de luminosité est d'obtenir une information de distance par rapport à la source lumineuse. Tout d'abord, les fenêtres sont comblées afin d'obscurcir la pièce. Les mesures de luminosité ne dépendent ainsi que de la source lumineuse. Il convient aussi d'éviter les reflets intempestifs et de choisir une source lumineuse adaptée à nos besoins. Les reflets sont éliminés par l'ajout de papier noir sur le mur bordant les tables. Cet ajout présente néanmoins un désavantage pour la communication infra-rouge qui sera abordé dans la partie communication.

Le test de plusieurs sources lumineuses telles qu'une lampe de poche, une lampe de bureau et des ampoules de puissance différente, nous a amené à choisir une ampoule de faible puissance. En effet, la lumière diffusée par une lampe de poche n'était pas assez uniforme. Une ampoule de puissance élevée (plus de 50 Watts) ne fournissait pas d'information lumineuse exploitable. Pour autant que le senseur se trouvait dans une zone éclairée, le niveau de luminosité était maximum, éliminant ainsi toute nuance entre les différentes positions du robot. Une ampoule de 25 Watts permet de pallier ces défauts.

La communication. Les limitations de la communication infra-rouge ont été abordés dans la section 4.3 traitant du protocole de communication. Pour l'environnement du laboratoire, trois tours infra-rouges sont nécessaires pour assurer une communication rapide avec un taux d'erreurs acceptable. En effet, le rebond d'ondes est limité par la présence de papier noir. Il en résulte une communication plus ardue entre une tour infra-rouge et le robot, dépendante de la direction de ce dernier.

6.1.2 Printemps des Sciences

L'environnement expérimental pour l'exposition du Printemps des Sciences consistait en un stand sous un chapiteau. Le degré de liberté était donc beaucoup plus limité. La tâche ne requérant pas de mesures de luminosité, il nous suffisait de fixer au sol une planche faisant office de surface dure et lisse, facilitant ainsi le déplacement du robot mobile, et de disposer plusieurs tours infra-rouges afin de rendre la communication fiable.

6.2 Téléguidage du Robot

Ce projet, réalisé en collaboration avec S. Collette, J. Hubert, D. Muquardt et A. Paszukiewicz, met en scène le robot mobile a des fins de téléguidage.

Le téléguidage se déroulait de la manière suivante :

- Deux webcams filment l'utilisateur tenant un objet de couleur spécifique. Cette couleur doit être unique au décor ainsi qu'à l'utilisateur. Une balle de couleur jaune est utilisée dans ce but. La figure 6.1 montre le stand et l'utilisateur manipulant la balle jaune.
- Le repérage et le suivi de l'objet se base sur la couleur caractéristique de l'objet. Une interface permet de choisir la couleur à tracer. Ensuite, un algorithme en calcule les coordonnées en trois dimensions.
- Un second algorithme est chargé de reconnaître les mouvements effectués par l'utilisateur sur base de ces coordonnées. Plusieurs formes sont définies comme un mouvement vers la gauche, la droite, en haut, en bas et un carré. Une fois la forme reconnue, une suite d'actions prédéfinie est réalisée en envoyant les ordres au robot.
- Les ordres sont communiqués au robot par le protocole de communication (voir section 4.3). Les mouvements sont gérés par le contrôle des moteurs développé dans le cadre de ce mémoire (voir section 4.2).

Ce projet fut choisi pour son interactivité et l'utilisation intéressante de plusieurs types de matériel tel que le robot et les webcams. Un projet futur pourrait facilement mettre en oeuvre un repérage/poursuite d'objet plus évolué, plusieurs robots ainsi que des actions plus complexes.

6.3 Recherche de la Lumière

Cette tâche de contrôle déjà décrite au chapitre 5, est mise en oeuvre par les expériences décrites ci-dessous.

Les performances des différentes politiques de contrôle sont déterminées par des critères arbitraires. La complétion de la tâche consiste à ce que le robot atteigne une limite proche de la source lumineuse à partir de sa position initiale. Une fois cette limite franchie, la recherche est considérée terminée. La figure 6.2 représente la recherche de la source lumineuse.

FIG. 6.1 – Vue du stand du Printemps des Sciences 2003 et l'utilisateur manipulant la balle jaune devant les deux webcams.

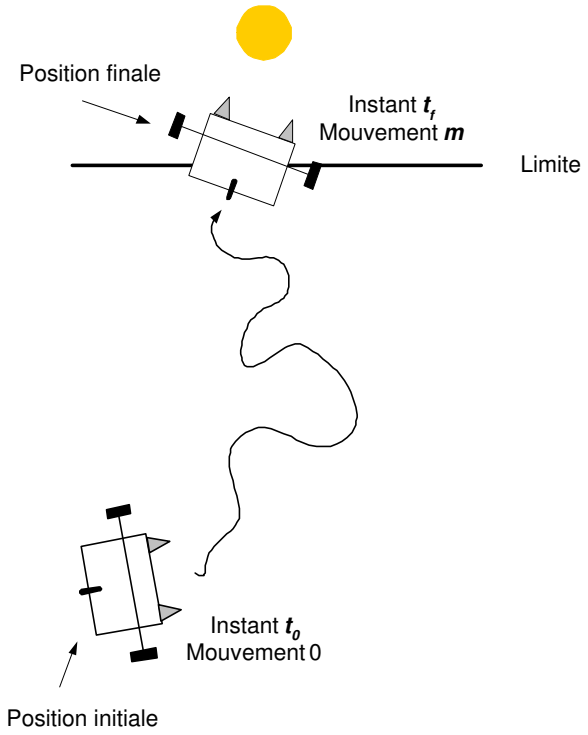


FIG. 6.2 – Recherche de la source lumineuse en considérant le temps et le nombre de mouvements effectués.

Les critères sélectionnés sont les suivants :

1. Le nombre de mouvements effectués pour la complétion de la tâche (m est le nombre de mouvements effectués sur la figure 6.2). Les prises de mesures ne sont pas prises en compte.
2. La durée totale de la complétion de la tâche ($t_f - t_0$ est la durée de l'expérience sur la figure 6.2).

Il reste à déterminer les points de départ pour la comparaison entre les politiques de contrôle. Deux points éloignés sont choisis et illustrés par la figure 6.3. Le choix de deux points de départ différents permet de tester les algorithmes de contrôle dans des situations différentes. En effet, le trajet que doit effectuer le robot à partir du *départ 1* est plus long mais plus direct pour être en face de la source lumineuse. Au contraire, le trajet partant du *départ 2* est plus court mais les changements de direction pour être en face de la source lumineuse, seront plus fréquents.

6.3.1 Politique de Contrôle Figée

L'expérience est réalisée grâce au code C donné en annexe J.1. Les résultats obtenus sont donnés par la table 6.1. Ils sont tirés d'une unique expérience représentative des différents tests effectués.

	Départ 1	Départ 2
Nombre d'actions	99	64
Durée de l'expérience (sec)	168	112

TAB. 6.1 – Résultats obtenus par l'implémentation de la politique de contrôle figée. Les colonnes spécifient le point de départ et les lignes spécifient le nombre d'action et la durée de l'expérience.

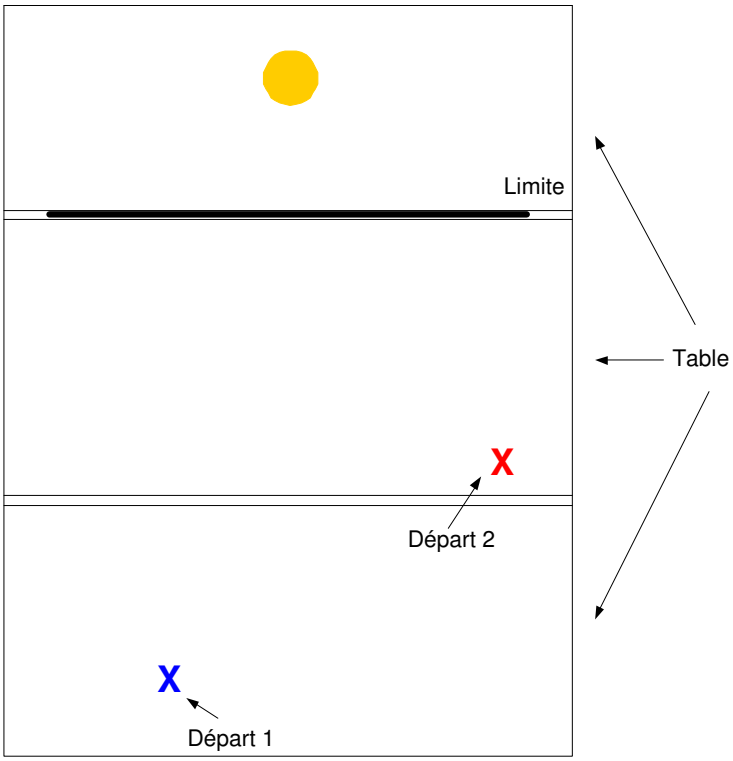


FIG. 6.3 – Points de départ choisis pour la comparaison entre les différentes politiques de contrôle.

Cette politique de contrôle est gourmande en nombre d'actions effectuées mais la durée totale de l'expérience est acceptable. Etant donné que cette politique est réalisée principalement à titre de comparaison, d'autres remarques sur les performances seront données par la suite.

6.3.2 Politique de Contrôle Basée sur l'Analyse de Données Sensorielles

L'expérience est réalisée grâce au code C donné en annexe J.2. Les résultats obtenus sont donnés par la table 6.2. Ils sont tirés d'une unique expérience représentative des différents tests effectués.

	Départ 1	Départ 2
Nombre d'actions	52	46
Durée de l'expérience (sec)	313	285

TAB. 6.2 – Résultats obtenus par l'implémentation de la politique de contrôle basée sur l'analyse des données sensorielles. Les colonnes spécifient le point de départ et les lignes spécifient le nombre d'action et la durée de l'expérience.

Cette politique de contrôle est plus économe que la précédente en terme de nombre d'actions effectuées. Dans le cas du trajet le plus long (à partir du départ 1, voir figure 6.3), un peu plus de la moitié des actions sont effectuées en comparaison avec la politique de contrôle figée. Cette différence est moins importante pour le trajet plus court.

La durée totale de l'expérience est importante à cause du nombre de mesures de luminosité effectuées à chaque action ainsi qu'aux calculs statistiques au niveau de l'ordinateur maître. La politique de contrôle basée sur l'analyse des données sensorielles peut s'avérer plus intéressante

si le coût des actions est élevé, dépense d'énergie et lenteur des moteurs par exemple. Dans le cadre de ce mémoire, le coût des actions est faible comparé à la durée de l'expérience, il est plus intéressant que le robot atteigne rapidement la source lumineuse.

6.3.3 Politique de Contrôle Basée sur l'Apprentissage de la Relation Sensori-Motrice

La mise en oeuvre de la politique de contrôle basée sur l'apprentissage de la relation sensori-motrice implique l'acquisition de données d'apprentissage, l'apprentissage du modèle et sa validation, ainsi qu'une mesure des performances du contrôle avec le robot.

6.3.3.1 Acquisition des Données

L'acquisition de données est un élément clé du plan expérimental (voir section 5.4). Il convient de collecter les données représentant le mieux possible le phénomène sous-jacent. Les données sont dites intéressantes si les mesures de luminosité portent sur la zone lumineuse. Etant donné que la zone obscure est plus étendue que la zone lumineuse (le cône de lumière émise par la source lumineuse est petit comparé à la globalité de l'environnement), de nombreuses mesures se font dans l'obscurité. Il convient donc de maximiser les mesures porteuses d'information lumineuse.

La modélisation est de type direct (voir section 5.4.1), les données d'apprentissage sont gardées sous la forme d'une base de données telle que le présente la figure 6.4 [7]. La modélisation ne

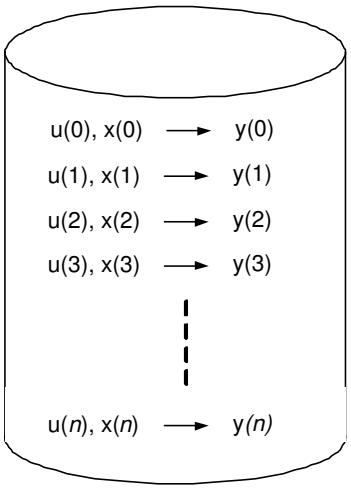


FIG. 6.4 – Base de données implémentant un modèle direct.

portant que sur les rotations (voir section 5.4.1), les différentes techniques d'acquisition de données effectueront des actions simples de type $[u_1, u_2]$ telle que $u_1 = -u_2$. A chaque prise de données, l'action, l'état antérieur et postérieur à cette action sont sauvegardés. L'ensemble des données d'apprentissage est formaté sous la forme structurée d'entrée/sortie donnée par la table 6.3.

Acquisition aléatoire. Cette première technique d'acquisition est la plus simple. Deux types de mouvements sont effectués : des déplacements et des rotations. Les actions sont tirées au hasard. Afin d'accélérer la prise de données, un nombre arbitraire[1] de rotations sont effectuées entre chaque déplacement. Rappelons que seules les rotations sont sauvegardées.

Cette technique permet de tester chaque action mais présente le désavantage de ne privilégier aucune direction particulière. Il en résulte de nombreuses mesures inintéressantes c'est-à-dire dans l'obscurité.

[1]Le nombre de rotations entre chaque déplacement est fixé à cinq.

u_1	u_2	x_1	x_2	v_1	v_2
0	0	546	657	554	678
-2	2	345	856	567	436
...	...	...	...	...	...

TAB. 6.3 – Format structuré d'entrée/sortie du fichier contenant l'ensemble des données d'apprentissage. u_1 et u_2 sont les actions appliquées respectivement sur m_1 et m_2, (x_1,x_2) et (v_1,v_2) sont les mesures de luminosité avant et après mouvement du robot.

Le pseudocode de l'algorithme d'acquisition de données aléatoire est donné par l'algorithme 4. Un pseudocode plus détaillé est donné en annexe H.1.

Algorithme 4 Pseudocode de l'algorithme pour l'acquisition de données aléatoire. Les prises de données sont sauvegardées sous le format spécifié en section 6.3.

```
FOR toutes les données à collecter
  IF phase de déplacement THEN
    Effectuer action [u1, u2] avec u1 = u2 tiré au hasard
  ELSE //Phase de rotation
    Effectuer action [u1, u2] avec u1 = −u2 tiré au hasard
  ENDIF
  Sauvegarder action et mesures lumineuses antérieures et postérieures à l'action
ENDFOR
```

Acquisition assistée. La technique d'acquisition assistée tente de limiter le principal désavantage de la technique précédente à savoir la majorité des mesures prises dans l'obscurité. Les mouvements ne se font plus au hasard mais sont le résultat d'une technique d'exploration similaire à la politique de contrôle figée (voir section 5.2). Les rotations sont tirées au hasard mais un repositionnement est effectué à la fin de chaque série. Ce repositionnement permet au robot de se diriger approximativement vers la source lumineuse. Un déplacement est ensuite appliqué. Nous obtenons donc des mesures pour les différentes actions de rotation tout en restant dans les environs du cône de lumière.

Le pseudocode de l'algorithme d'acquisition de données assistée est donné par l'algorithme 5. Un pseudocode plus détaillé est donné en annexe H.2.

Algorithme 5 Pseudocode de l'algorithme pour l'acquisition de données assistée. Les prises de données sont sauvegardées sous le format spécifié en section 6.3.

```
FOR toutes les données à collecter
  IF phase de déplacement THEN
    Effectuer rotation de repositionnement vers la source lumineuse
    Effectuer action [u1, u2] avec u1 = u2 tiré au hasard
  ELSE //Phase de rotation
    Effectuer action [u1, u2] avec u1 = −u2 tiré au hasard
    Enregistrer la rotation vers la source lumineuse pour le repositionnement
  ENDIF
  Sauvegarder action et mesures lumineuses antérieures et postérieures à l'action
ENDFOR
```

Acquisition par rotation. Cette dernière technique permet à l'utilisateur de collecter des données pour des rotations spécifiques. Elle est notamment utilisée pour les expériences 5.16, 5.10

et 5.13. L'utilisateur spécifie simplement l'amplitude de la rotation, le nombre de mesures à chaque rotation ainsi que le nombre d'itérations à effectuer.

Représentation des données collectées. Une représentation des données collectées permet d'estimer leur qualité. Dans le cas idéal, il faudrait collecter des données en testant toutes les actions de rotation avec la même probabilité et que les mesures de luminosité antérieures et postérieures aux actions soient porteuses d'information lumineuse.

L'histogramme donné par la figure 6.5 représente les actions de rotation testées qui sont limitées de -5 à +5. Nous pouvons en conclure que toutes les actions ont été testées avec la même probabilité.

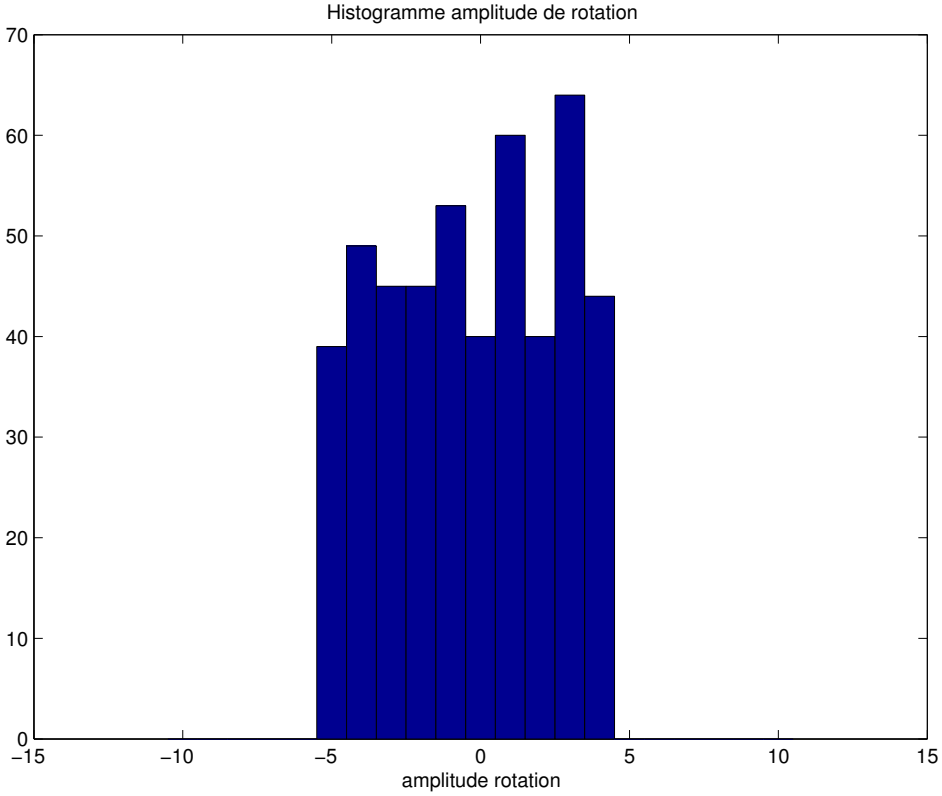


FIG. 6.5 – Histogramme représentant les actions de rotations utilisées pour la collecte des mesures lumineuses. Les amplitudes des actions de rotation sont limitées de -5 à +5.

Il nous faut à présent estimer la qualité des mesures de luminosité. En premier lieu, les histogrammes donnés par la figure 6.6 permettent d'estimer la proportion de mesures porteuses d'information lumineuse et ce, pour chaque senseur indépendamment.

Les informations mises en évidence par les deux histogrammes donnés par la figure 6.6 ne donnent pas l'information de dépendance entre les deux senseurs. Des mesures de luminosité intéressantes sont constituées de trois catégories de couples de mesures (x_1, x_2) :

1. (luminosité importante, luminosité faible). Intuitivement, la direction du robot est située vers la droite de la source lumineuse.
2. (luminosité faible, luminosité importante). Intuitivement, la direction du robot est située vers la gauche de la source lumineuse.
3. (luminosité importante, luminosité importante). Intuitivement, le robot est situé en face de la source lumineuse.

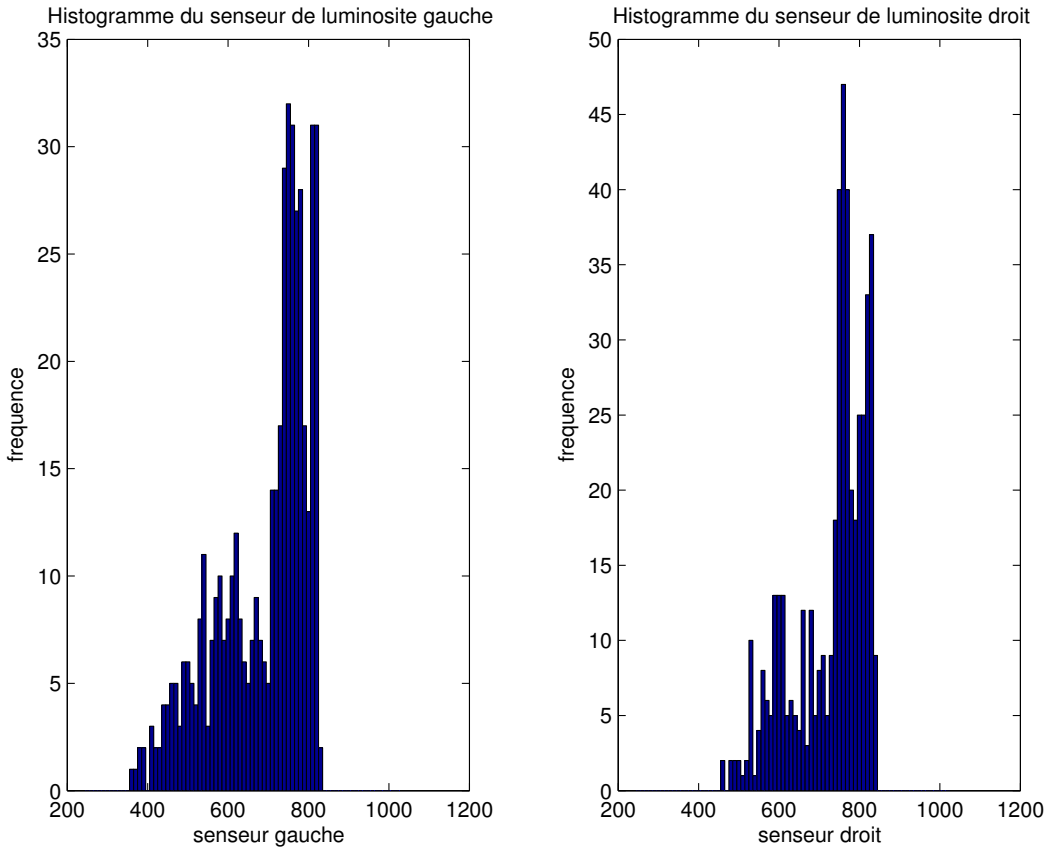


FIG. 6.6 – Histogrammes représentant les mesures de luminosité collectées après action pour les deux senseurs.

Le couple (luminosité faible, luminosité faible) étant une mesure dans l'obscurité et donc inintéressante. la figure 6.7 représente les positions correspondant aux différents couples.

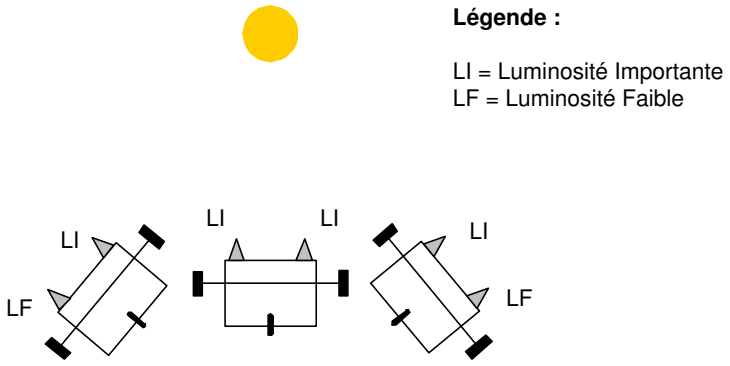


FIG. 6.7 – Positions intuitives pour les différents couples de mesures.

Un bon ensemble de données d'apprentissage doit donc contenir ces trois catégories de mesure. La figure 6.8 représente la fréquence des couples de mesures de luminosité.

Nous pouvons remarquer que la majorité des mesures se font dans l'obscurité malgré les techniques d'acquisition qui limitent ce phénomène. il est aisé de compenser cette tendance en prenant manuellement des mesures proche de la source lumineuse. Cependant, les données collectées sans compensation manuelle sont suffisantes pour les fins de modélisation prévues.

6.3.3.2 Validation de Modèle

La validation des modèles linéaire et non-linéaire permettra de juger de la qualité de ceux-ci avant leur mise en pratique dans la politique de contrôle basée sur l'apprentissage de la relation sensori-motrice.

Modèle linéaire. Nous utiliserons MATLAB pour la validation du modèle linéaire. A partir de l'ensemble de données original, le modèle est construit et sa faculté de généralisation calculée par l'estimation de l'erreur de généralisation par la technique *L-O-O* (voir section 5.4.1.3). Le script MATLAB *valid_linear_model.m*[2] calcule cette estimation.

Voici les résultats obtenus avec le modèle linéaire sur un ensemble de 300 données d'apprentissage : $\hat{G}_N^1 = 5279,4$ et $\hat{G}_N^2 = 3852,1$ avec $\hat{G}_N^1$ et $\hat{G}_N^2$, les erreurs de généralisation pour le modèle du senseur gauche et droit. Nous comparerons ces résultats avec ceux du modèle non-linéaire, c'est-à-dire le Lazy Learning.

Afin de mieux appréhender l'importance de l'ensemble des données d'apprentissage, une comparaison entre le nombre de données utilisées pour l'apprentissage du modèle et sa performance en terme d'estimation de l'erreur de généralisation, est effectuée par le même script MATLAB. La figure 6.9 montre les résultats de cette comparaison sous forme de graphe.

L'ajout de nouvelles données améliorent progressivement les performances modèle linéaire de la relation sensori-motrice mais le gain de performances n'est pas aussi significatif que pour le Lazy Learning comme nous le verrons ci-dessous. Remarquons que les performances du modèle linéaire se dégradent à partir d'un certain nombre de données d'apprentissage. Cette observation est un indice de non-linéarité du phénomène sous-jacent les données.

Modèle non-linéaire : Lazy Learning. Les analyses sont identiques à celles effectuées pour le modèle linéaire. Le script MATLAB *valid_lazy_model.m*[3] calcule l'estimation de l'erreur de généralisation du modèle construit grâce au Lazy Learning.

[2]Le code du script *valid_linear_model.m* est donné en annexe I.1.
[3]Le code du script *valid_linear_model.m* est donné en annexe I.2.

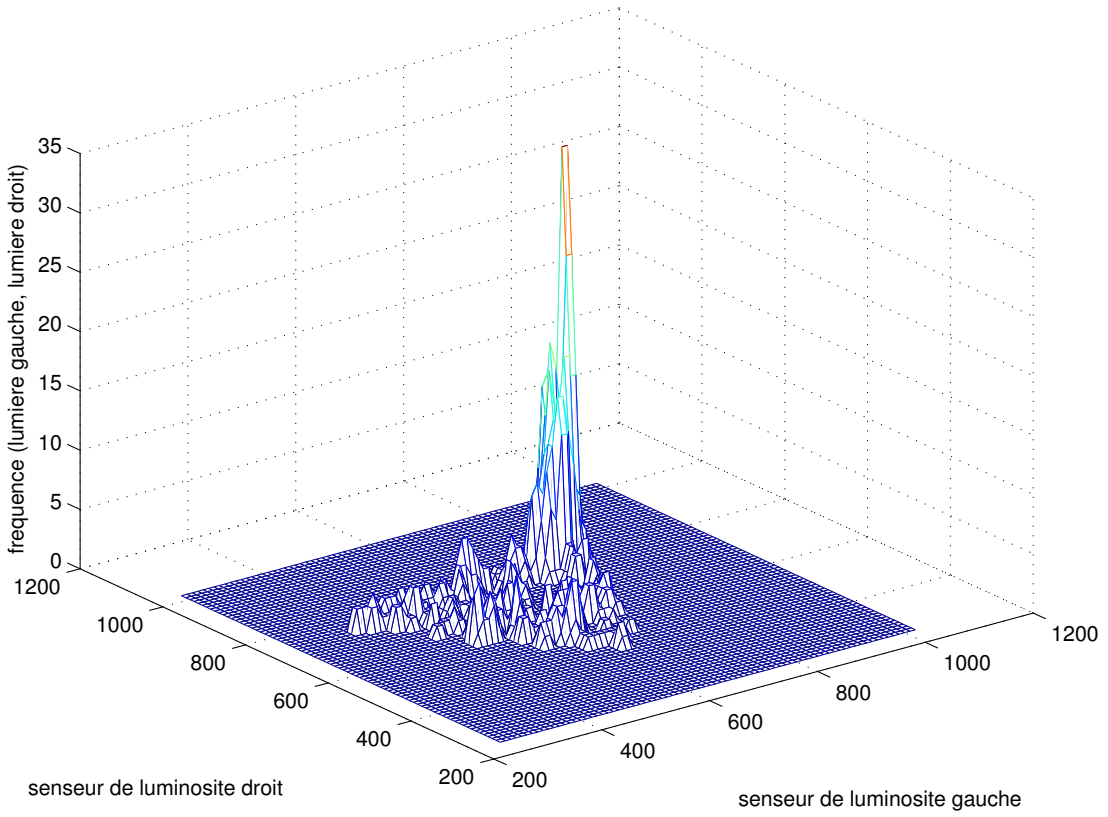


FIG. 6.8 – Histogramme représentant les couples de mesures de luminosité collectées après action pour les deux senseurs.

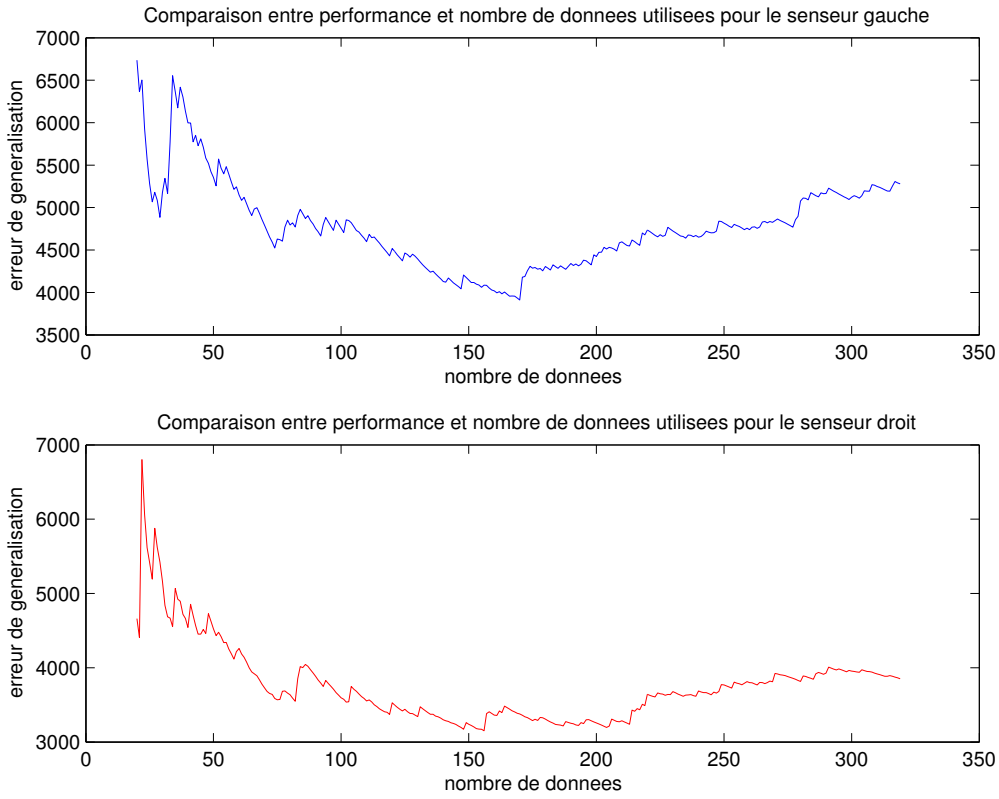


FIG. 6.9 – Graphe comparant le nombre d'observations utilisées lors de la construction du modèle linéaire et sa précision.

Voici les résultats obtenus avec le modèle Lazy Learning et le même ensemble de 300 données d'apprentissage : $\hat{G}_N^1 = 3565,5$ et $\hat{G}_N^2 = 3289,3$.

Le même script MATLAB effectue la comparaison entre le nombre d'observations utilisées pour la construction du modèle et sa performance. La figure 6.9 montre les résultats de cette comparaison sous forme de graphe.

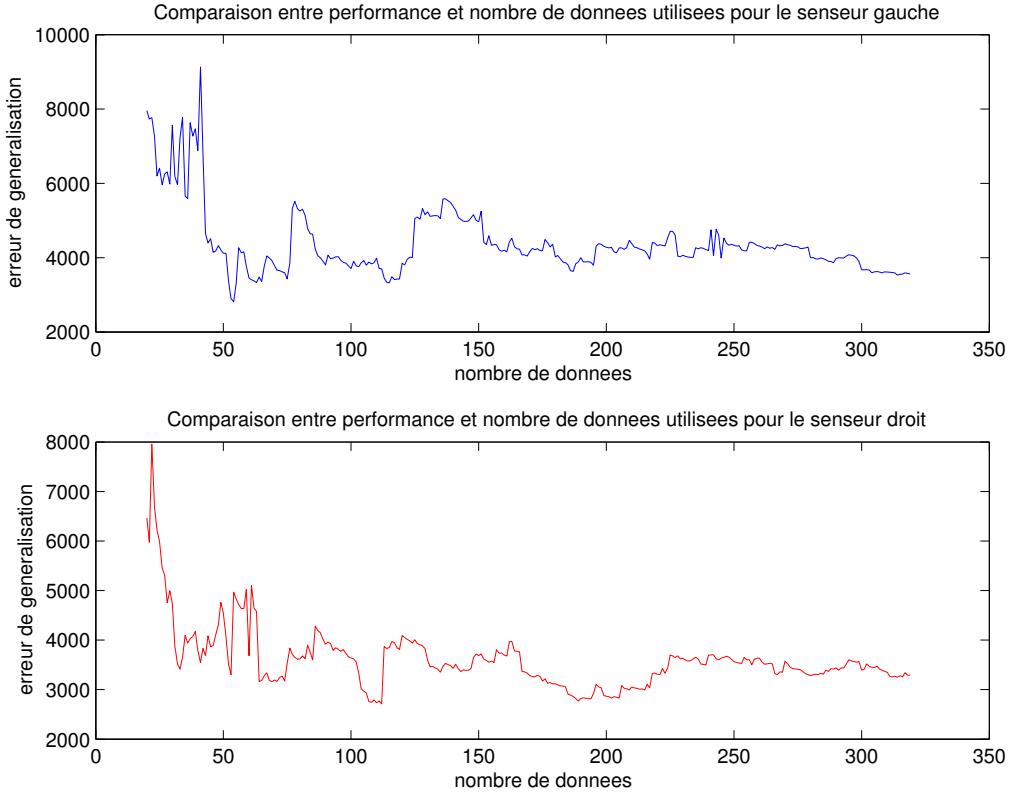


FIG. 6.10 – Graphe comparant le nombre d'observations utilisées lors de la construction du modèle par le Lazy Learning et sa précision.

L'ajout de données améliorent les performances plus rapidement que pour le modèle linéaire. De plus, il n'y pas de dégradation dû à l'ajout de nouvelles données au contraire du modèle linéaire car le phénomène est vraisemblablement non-linéaire.

Analyse comparative. La comparaison entre les estimations des erreurs de généralisation donne un net avantage au Lazy Learning. Qui plus est, l'ajout de nouvelles données améliorent toujours les performances du Lazy Learning ce qui s'avère intéressant pour des politiques de contrôle adaptatif.

Afin de mieux représenter la qualité des deux modèles étudiés, la figure 6.11 montre un exemple de prédictions pour les senseurs gauche et droit. A nouveau, nous pouvons remarquer que la faculté de prédiction du Lazy Learning est supérieure au modèle linéaire.

6.3.3.3 Performance de la Politique de Contrôle

Présentons à présent les résultats obtenus par la politique de contrôle basée sur l'apprentissage de la relation sensori-motrice, utilisant un modèle linéaire et non-linéaire.

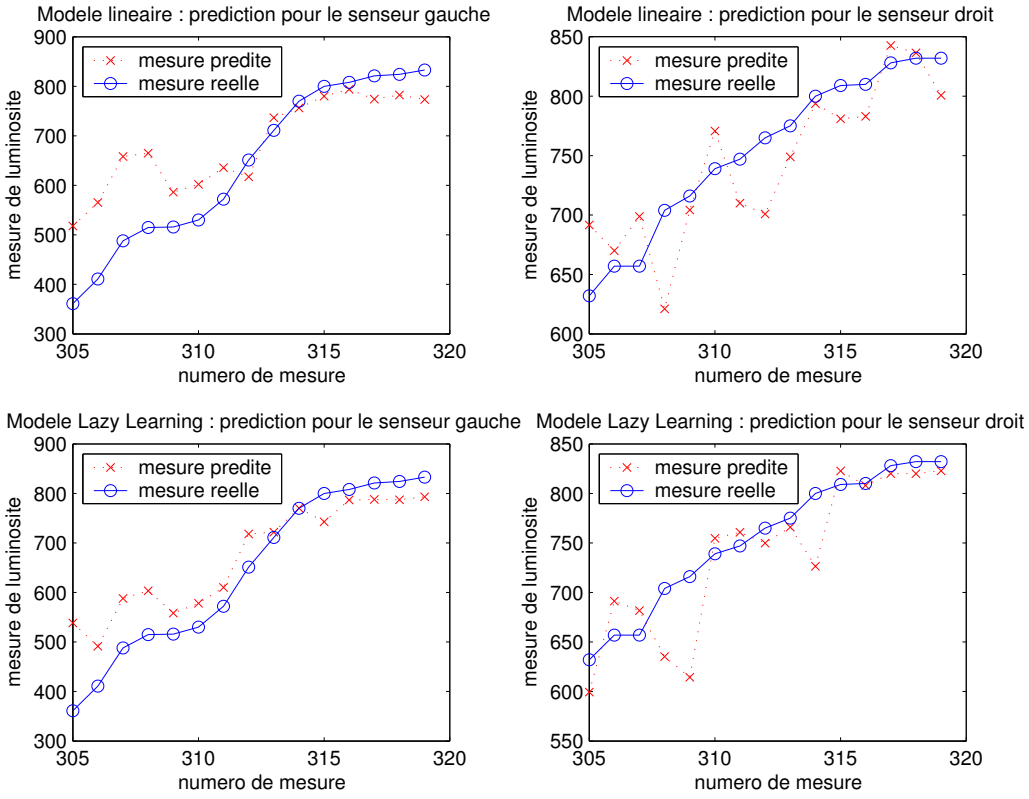


FIG. 6.11 – Comparaison entre les modèles Lazy Learning et linéaire. Une fois l'apprentissage du modèle terminé, 15 nouvelles données sont prédites. A titre de comparaison, les données réelles et prédites sont tracées pour les modèles linéaire et Lazy Learning et ce, pour les deux senseurs.

Après l'apprentissage du modèle, une fonction coût doit être choisie (voir section 5.4.2, équation 5.8). La fonction coût est importante car elle influence en partie les performances de l'algorithme de contrôle.

La tâche de contrôle consistant en la recherche d'une source lumineuse, il convient de choisir une fonction coût favorisant la position du robot en face de la lumière. L'analyse des données sensorielles (voir section 5.3.2) a mis en évidence l'observation suivante : la position du robot en face de la source lumineuse est caractérisée par une différence de luminosité entre l_1 et l_2 proche de zéro dans la zone lumineuse. Cette observation guidera le choix d'une fonction coût efficace pour la réalisation de la tâche de contrôle.

Le coût doit être minimal lorsque le robot est en face de la source lumineuse. La fonction coût choisie est du type

$$\begin{aligned} C &= c\,(x_1(k),\, x_2(k),\, u_1(k),\, u_2(k),\, \hat{x}_1(k+1),\, \hat{x}_2(k+1)) \\ &= |\hat{x}_1(k+1) - \hat{x}_2(k+1)| + dark \end{aligned} \tag{6.1}$$

avec

$$dark = \begin{cases} 0 & si\ \hat{x}_1(k+1) < 750 \vee \hat{x}_2(k+1) < 750 \\ 2048 & sinon \end{cases}$$

La fonction C est donc minimale lorsque la valeur absolue de la différence de luminosité entre l_1 et l_2 est nulle et ce, lorsque les mesures contiennent une information lumineuse[4]. La valeur *dark* permet de pénaliser les petites différences de luminosité dans l'obscurité ce qui évitera que le robot ne se dirige systématiquement vers une zone obscure.

Modèle linéaire. L'expérience est réalisée grâce au code C donné en annexe J.3.

Rappelons qu'après estimation des paramètres $\hat{\beta}_i$ par la technique *LS*, nous obtenons un modèle du type

$$\hat{x}(k+1) = \hat{\beta}_0 + \hat{\beta}_1 x_1(k) + \hat{\beta}_2 x_2(k) + \hat{\beta}_3 u_1(k) + \hat{\beta}_4 u_2(k)$$

Bien que la faculté de prédiction d'un tel modèle soit acceptable (voir section 6.3.3.2), il est inutilisable en pratique à cause du choix même de la fonction de coût. En effet, celle-ci privilégie en pratique des prédictions de mesures de luminosité $\hat{x}_1(k+1)$ et $\hat{x}_2(k+1)$ minimales afin que leur différence soit petite dans la zone lumineuse. Il en découle que le choix de l'action de rotation optimale sera toujours extrême et dépendra des paramètres estimés $\hat{\beta}_i$. Illustrons cette observation par un exemple de cas de figure hypothétique :

Soit $\hat{\beta}_0^1 = 10$, $\hat{\beta}_1^1 = 0,5$, $\hat{\beta}_2^1 = 0,001$, $\hat{\beta}_3^1 = 0,9$, $\hat{\beta}_4^1 = 0,6$, paramètres estimés du modèle linéaire de x_1.

Soit $\hat{\beta}_0^2 = 11$, $\hat{\beta}_1^2 = 0,6$, $\hat{\beta}_2^2 = 0,003$, $\hat{\beta}_3^2 = 0,4$, $\hat{\beta}_4^2 = 0,7$, paramètres estimés du modèle linéaire de x_2.

Soit U l'ensemble des actions de rotations allant de $[-5,\, 5]$ à $[5,\, -5]$.

L'action optimale est l'action de rotation $[-5,\, 5]$ car elle minimise les prédictions $\hat{x}_1(k+1)$ et $\hat{x}_2(k+1)$ et ce, quelque soit l'état du système $x(k)$ qui est constitué du couple de mesures de luminosité $(x_1(k),\, x_2(k))$. Il en résulte le choix systématique de l'action $[-5,\, 5]$ ce qui ne permet pas le contrôle du robot pour la recherche de la source lumineuse.

Modèle non-linéaire : Lazy Learning. L'expérience est réalisée grâce au code C donné en annexe J.3. Les résultats obtenus sont donnés par la table 6.4.

Dans le but d'augmenter le volume des données d'apprentissage, celles-ci sont enregistrées dans le fichier après chaque action de rotation. Ces nouvelles données ne seront utilisées que lors d'un nouveau lancement de l'algorithme de contrôle, l'ajustement du modèle ne s'effectuant qu'une seule fois. Nous verrons dans la section 5.4.4 traitant du contrôle adaptatif, que ces données pourraient être utilisées directement.

[4]Une mesure de luminosité est porteuse d'information lumineuse si sa valeur est inférieure à 750. Cette valeur peut varier selon les conditions expérimentales.

	Départ 1	Départ 2
Nombre d'actions	$23,6 \pm 2.2296$	$16,5333 \pm 1,5522$
Durée de l'expérience (sec)	$43,6667 \pm 3,8483$	$31,6 \pm 3,1577$

TAB. 6.4 – Résultats obtenus par l'implémentation de la politique de contrôle basée sur l'apprentissage de la relation sensori-motrice utilisant un modèle Lazy Learning. Les colonnes spécifient le point de départ et les lignes spécifient le nombre d'action et la durée de l'expérience.

Au contraire des autres politiques, quinze expériences ont été effectuées afin d'obtenir des résultats plus précis. La moyenne et la variance de chaque critère sont calculées. Remarquons la faible variance des critères observées surtout pour le trajet partant du départ 2 car celui-ci est plus court.

Cette politique de contrôle s'avère être la meilleure au regard des résultats obtenus. Le nombre d'actions est réduit de moitié par rapport à la politique de contrôle basée sur l'analyse de données sensorielles et vaut le quart de celui de la politique de contrôle figée. L'utilisation du modèle permet de n'effectuer qu'une seule action de rotation afin de diriger le robot vers la source lumineuse au contraire des autres politiques de contrôle. Cette économie importante d'actions réduit ainsi la durée de l'expérience.

La table 6.5 est un récapitulatif des performances pour les trois politiques de contrôle étudiées. Remarquons que la *PAR* avec un modèle Lazy Learning est la plus performante à tous les niveaux.

	Départ 1			Départ 2		
	PF	*PAS*	*PAR : LL*	*PF*	*PAS*	*PAR : LL*
Nombre d'actions	99	52	**23,6**	64	46	**16,5333**
Durée de l'expérience	168	313	**43,6667**	112	285	**31,6**

TAB. 6.5 – Comparaison des résultats obtenus par les trois implémentations des politiques de contrôle.

6.3.4 Politique d'Apprentissage Adaptatif du Contrôle

Afin d'évaluer l'adaptation du contrôle, deux tests sont effectués : l'un montrant l'utilité de l'ajout de nouvelles observations durant le contrôle, et l'autre mettant en avant l'adaptation au changement de la position des senseurs de luminosité du robot.

6.3.4.1 Ajout d'Observations en Cours de Contrôle

Nous montrons le gain de performances lors de la mise à jour du modèle par l'ajout des observations durant le contrôle. Nous avons expliqué en section 5.4.4, la facilité de mise à jour du modèle Lazy Learning. Pour mettre en évidence le gain de performances généré, l'expérience suivante est réalisée : l'algorithme de la *PAR* est mis en oeuvre avec un ensemble réduit de données d'apprentissage[5]. A chaque nouvelle action, le modèle est mis à jour par l'ajout de la nouvelle observation jusqu'à tripler le nombre total d'observations. La figure 6.12 montre l'évolution du nombre d'actions effectuées ainsi que de la durée de réalisation de la tâche de contrôle pour une série d'expériences consécutives. Une expérience est considérée terminée sous les mêmes critères que précédemment (voir section 6.3). Nous pouvons constater une amélioration progressive des performances. Une forte amélioration se produit lors de l'apprentissage des nouvelles observations durant les premières expériences pour ensuite varier légèrement.

[5]Seules 50 observations constituent l'ensemble de données d'apprentissage.

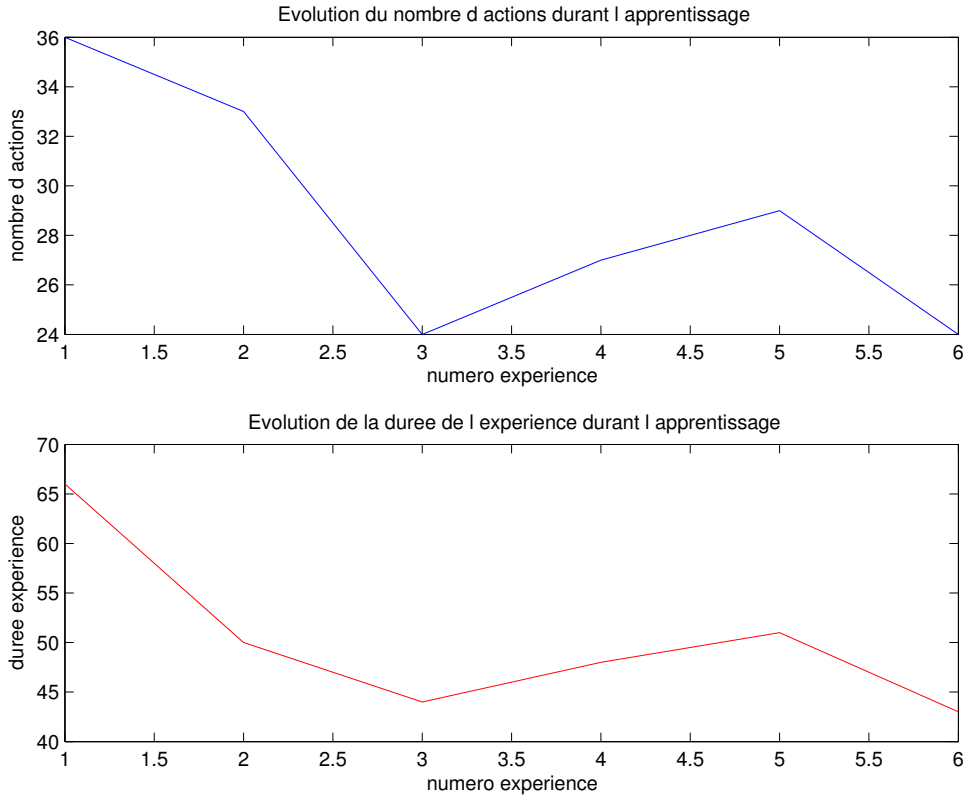


FIG. 6.12 – Evolution du nombre d'actions effectuées ainsi que de la durée de réalisation de la tâche de contrôle avec un ensemble de données d'apprentissage réduit et une mise à jour systématique du modèle. L'axe des abscisses représente le numéro des expériences, celles-ci se déroulant consécutivement.

L'ajout des observations en cours de contrôle améliore donc significativement les performances surtout si l'ensemble des données d'apprentissage est réduit.

6.3.4.2 Changement de Position des Senseurs de Luminosité

Considérons à présent un changement dans la configuration du robot : les senseurs de luminosité gauche et droit sont permutés. L'ensemble des données d'apprentissage sont acquises avec la configuration de base. Une fois les senseurs inversés, l'algorithme de contrôle est mis en oeuvre. Les données d'apprentissage ne sont donc pas représentatives de la configuration réelle du robot. Pour mettre en évidence l'adaptation du contrôle à la modification du robot, l'expérience suivante est réalisée : l'algorithme de la *PAR* est mis en oeuvre avec un ensemble réduit de données d'apprentissage[6] ainsi qu'un oubli progressif[7] des anciennes observations. Nous nous attendons à observer de mauvaises performances lors des premières expériences pour ensuite s'améliorer une fois les anciennes observations oubliées. La figure 6.13 montre l'évolution du nombre d'actions effectuées ainsi que de la durée de réalisation de la tâche de contrôle pour une série d'expériences consécutives.

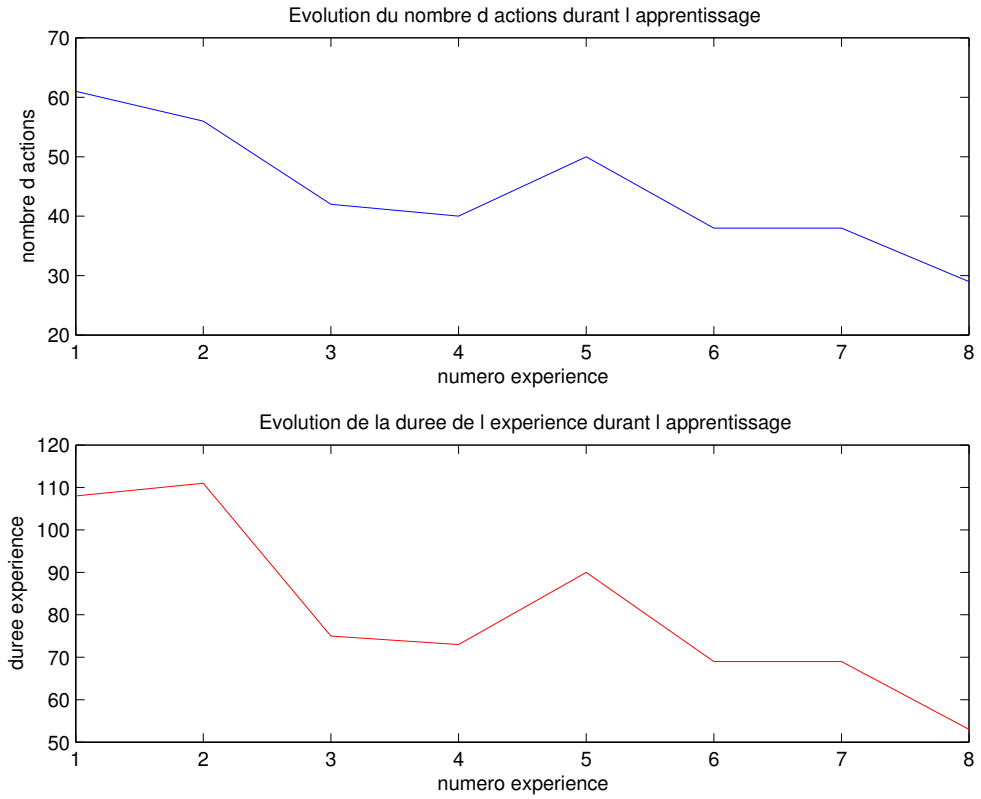


FIG. 6.13 – Evolution du nombre d'actions effectuées ainsi que de la durée de réalisation de la tâche de contrôle avec un ensemble de données d'apprentissage réduit et une mise à jour systématique du modèle. Les senseurs de luminosité ont été préalablement inversés. L'axe des abscisses représente le numéro des expériences, celles-ci se déroulant consécutivement.

Les expériences confirment nos attentes avec une forte amélioration des performances pour les premières expériences. A partir de la quatrième expérience, les observations acquises avec l'an-

[6] Seules 50 observations constituent l'ensemble de données d'apprentissage.

[7] Un facteur d'oubli de trois est utilisé lors des expériences, signifiant que la donnée la plus ancienne est effacée lorsque trois nouvelles observations sont mémorisées.

cienne configuration du robot sont totalement oubliées, et seules des observations représentatives de la configuration réelle sont utilisées.

6.3.5 Films

Les expériences concernant les politiques de contrôle ont été filmées afin de mieux illustrer les résultats obtenus. Les films ainsi réalisés mettent en évidence le manque de fluidité dans le comportement du robot. Cet inconvénient provient principalement des senseurs de luminosité qui imposent une pause importante entre chaque action du robot. En effet, les délais de communication et de prédiction par le Lazy Learning sont négligeables face aux délais nécessaires pour les mesures de luminosité.

Chapitre 7

Conclusion

La robotique autonome étant un domaine de recherche interdisciplinaire, ce mémoire demanda le développement de compétences tant matérielles et logicielles. La plateforme robotique développée a permis la réalisation de la tâche de contrôle choisie et de mettre en oeuvre les différentes politiques de contrôle étudiées. Les LEGO Mindstorms, bien que limités, s'avère un excellent outil pour introduire les concepts de base de la robotique. Le protocole de communication développé fut une étape déterminante dans l'exploitation de cette plateforme : la puissance de calcul ainsi que la mémoire disponible ne sont plus limités par le RCX qui se cantonne à l'exécution d'ordres simples. Nous disposons maintenant de la puissance d'un ordinateur afin de contrôler le robot mobile.

Cette plateforme est particulièrement adaptée aux projets éducatifs par sa simplicité et son potentiel comme en témoigne le téléguidage du robot lors du Printemps des Sciences.

Les expériences ont mis en évidence la supériorité de la politique de contrôle basée sur la relation sensori-motrice sur les politiques de contrôle basée uniquement sur les connaissances du concepteur. En effet, aucune supposition ne peut être faite sur le bruit des données sensorielles et l'imprécision des mouvements du robot ce qui complique considérablement le travail d'un concepteur travaillant selon une approche classique. La méthode d'apprentissage s'avère une alternative augmentant significativement les performances du contrôle.

L'adaptation de la politique de contrôle basée sur l'apprentissage de la relation sensori-motrice a permis la mise en oeuvre d'expériences mettant en évidence le gain apporté par la méthode d'apprentissage local. La partie adaptative, bien qu'à peine abordée dans le cadre de ce mémoire, apparaît très prometteuse et peut faire l'objet de travaux futurs intéressants. Cette conclusion rejoint les remarques présentées dans l'état de l'art : la complexité des robots mobiles autonomes et leur environnement, rend la tâche des concepteurs extrêmement ardue pour l'approche classique.

7.1 Travaux Futurs

Les expériences effectuées dans le cadre de ce mémoire n'utilisent qu'une partie des possibilités offertes par la plateforme développée. De nombreux travaux futurs pourraient être menés à bien dont quelques exemples sont donnés ci-dessous.

Utilisation d'autres méthodes d'apprentissage. La tâche considérée est principalement réactive : les mesures de luminosité après action fournissent directement au robot une information qualitative de l'action effectuée. C'est la raison pour laquelle une méthode d'apprentissage supervisé fut utilisée. Le choix d'une tâche plus complexe pourrait mettre en oeuvre une méthode d'apprentissage non supervisée ou par renforcement [35, 34, 33]. Citons par exemple, le parcours d'un labyrinthe, exemple de prédilection de la mise en oeuvre d'une technique de type *Q-learning* [85].

Modélisation pour la recherche de la source lumineuse. Le modèle sélectionné pour la politique de contrôle basée sur l'apprentissage de la relation sensori-motrice est limité aux actions de rotations du robot afin de simplifier l'analyse du comportement du robot. Cependant, il pourrait être plus efficace de considérer l'ensemble des actions ce qui rend l'analyse de la relation sensori-motrice plus complexe.

Poursuite de la source lumineuse. La source lumineuse pourrait être manipulée par un utilisateur qui observerait le robot suivre ses mouvements. Une autre tâche de contrôle envisageable serait la poursuite d'un autre robot mobile sur lequel serait fixée une source lumineuse. Cela donnerait lieu à des expériences visuellement attractives.

Mise à disposition publique du code source. Les outils pour LEGO Mindstorms sont généralement le résultat de projets *open-source*. La mise à disposition publique est primordiale pour l'évolution de la plateforme LEGO Mindstorms. Les codes sources qui pourraient être mis à disposition publique, sont ceux de la gestion du matériel au niveau du robot, du protocole de communication et des politiques de contrôle.

Adaptation de la politique de contrôle basée sur la relation sensori-motrice. Seuls deux types d'adaptation ont fait l'objet d'expériences : l'adaptation réalisée grâce à l'arrivée de nouvelles données en cours de contrôle et l'adaptation au changement de la position des senseurs. Il existe une multitude d'autres facteurs qui peuvent être modifiés tels que la cinématique du robot, sa taille ainsi que son poids et l'amplitude des actions par exemple.

Tâches de contrôle. Bien que limitée, la plateforme robotique LEGO Mindstorms offre de nombreuses possibilités pour la réalisation d'autres tâches que celle étudiée. Nous pourrions complexifier la structure du robot et rajouter d'autres senseurs.

Contrôle de plusieurs robots. Le protocole de communication développé dans le cadre de ce mémoire, permet le contrôle de plusieurs robots ce qui s'avérerait intéressant pour d'autres expériences similaires au projet *swarm-bots* [53] ou le projet [81].

Plateforme éducative. La plateforme robotique LEGO Mindstorms permet la réalisation de projets pour des cours universitaires tels que *Modèles Stochastiques II* [17] donné à l'ULB par G. Bontempi.

Annexe A

RCX

A.1 Description des Composants

Le RCX n'est pas constitué d'une seule et unique puce mais de l'assemblage de plusieurs composants d'origines différentes. Une liste des composants et périphériques principaux est donnée ci-dessous. La figure A.1 montre le RCX après démontage de la boîte LEGO externe.

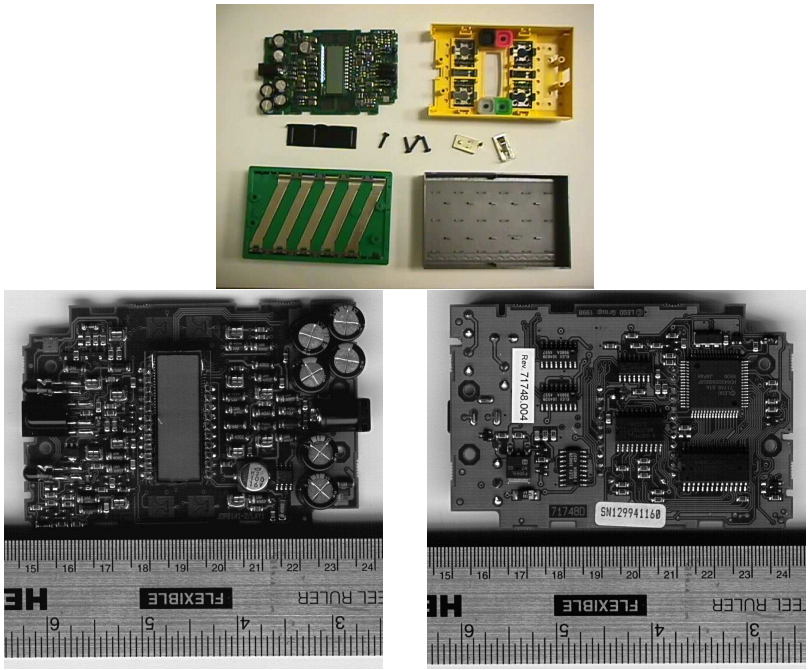


FIG. A.1 – RCX après démontage.

A.1.1 Le Microcontrôleur

La microcontrôleur embarqué dans le RCX est l'HITACHI H8/3292 de la famille des H8/3297. Cette puce intègre un processeur de type H8/300 ainsi que la mémoire et les entrées/sorties. Ces composants internes sont connectés par 2 bus de données 8-bit bidirectionnels (Data Bus Low et Data Bus High) ainsi qu'un bus 16-bit d'adressage. Le schéma bloc du H8/3292 est donné par la figure A.2 et l'arrangement des pattes par la figure A.3.

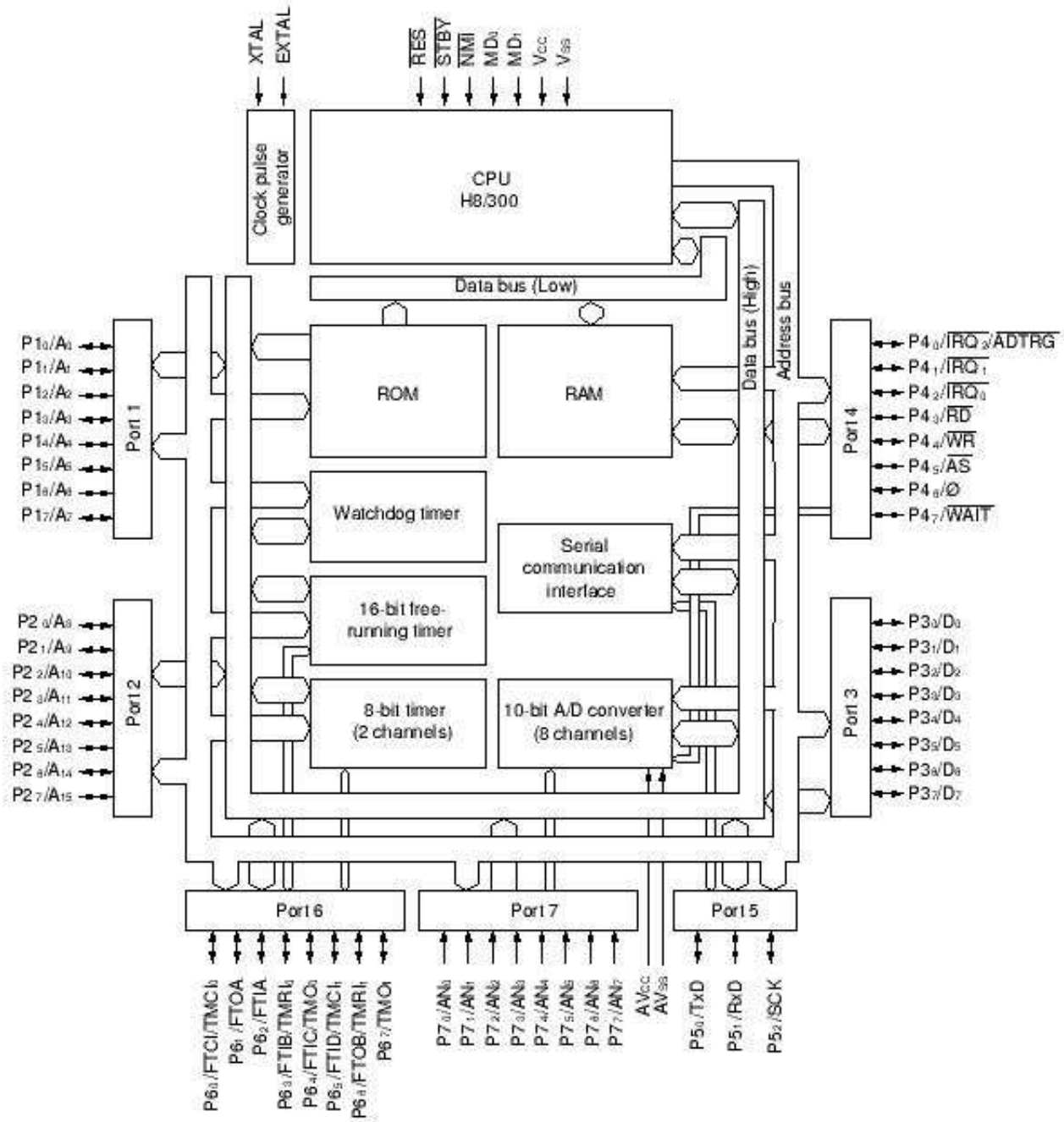


FIG. A.2 – Schéma bloc de la famille des H8/3297 (figure tirée du manuel [47]).

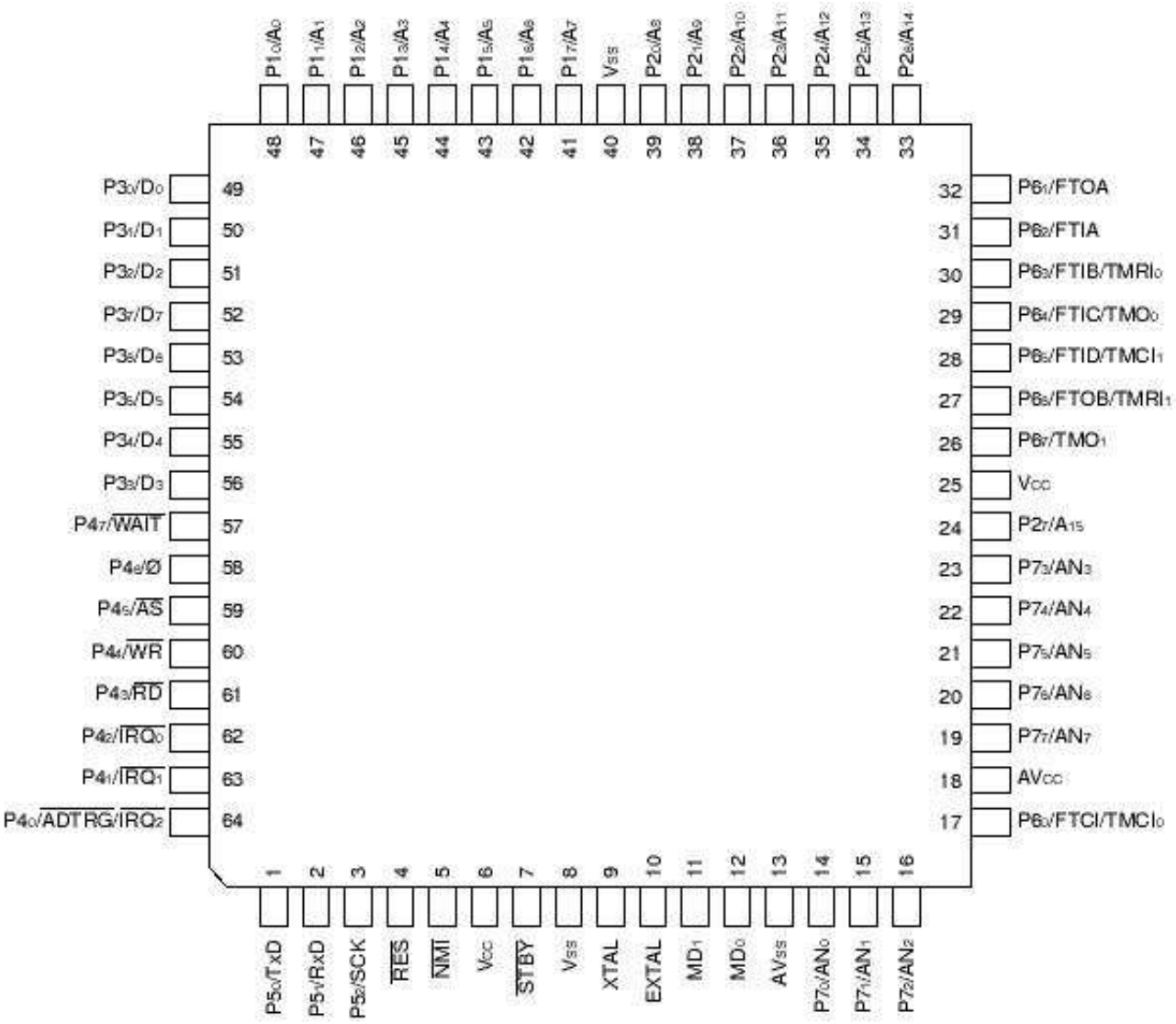


FIG. A.3 – Arrangement des pattes du H8/3292 (figure tirée du manuel [47]).

A.1.1.1 Le Processeur

Le processeur est donc le H8/300 [46], processeur 8-bit, basé sur une architecture à registres généraux. Il s'agit d'une architecture orthogonale où toutes les opérations sont effectuées entre registres. Les contenus des registres peuvent être bougés, chargés ou stockés librement. Il dispose de 8 registres de 16 bits qui sont aussi configurables en 16 registres de 8 bits.

L'ensemble des instructions en est composé de 57 conçues pour des opérations rapides. Bien que le H8/300 soit un processeur CISC, il possède quelques éléments de l'architecture RISC comme de nombreux processeurs modernes :

- Son jeu d'instructions est concis (57 instructions).
- Les opérations arithmétiques et logiques ne portent que sur des registres ou des données immédiates. On peut parler d'architecture de type *load and store* (les données sont préalablement chargées dans des registres avant opérations).
- Instructions de multiplication et division rapides. De plus, une série d'instructions sont spécialisées dans la manipulation de bits.
- La plupart des instructions tiennent en 2 bytes (maximum 4) et peuvent être exécutées en 2 états du processeur.
- 8 modes d'adressage.

Spécifications

- Double configuration des registres : 8 ou 16 bits.
- Ensemble de 57 instructions dont :
  - Opérations de division et multiplication.
  - Instructions puissantes sur les manipulations de bits.
- 8 modes d'adressage[1] dont :
  - Mode d'adressage direct (Rn).
  - Mode d'adressage indirect (@Rn).
  - Mode d'adressage indirect avec déplacement (@(d :16,Rn)).
  - Mode d'adressage indirect avec post-incrémentation ou pré-décrémentation (@Rn+ ou @-Rn).
  - Adresse absolue (@aa :8 ou @aa :16).
  - Adresse immédiate (#xx :8 ou #xx :16).
  - Adresse relative au PC (@d :8,PC).
  - Mémoire indirecte (@@aa :8).
- Espace maximum d'adressage de 64 Kbytes (adresses de 16 bits) pour le programme et ses données.
- Vitesse d'horloge de 16 MHz à 5V.
- Opérations rapides :
  - Soustraction ou addition sur registres de 8 ou 16 bits en 125 ns.
  - Multiplication de registres de 8 bits en 875 ns.
  - Division d'un registre de 16 bits par un registre de 8 bits en 875 ns.
- Mode de mise en veille grâce à l'instruction *sleep*.

A.1.1.2 La Mémoire

La mémoire intégrée consiste en 16 Kbytes de mask ROM[2] et en 512 bytes de RAM. En outre, il existe un champ de registres intégrés (dont la taille est de 128 bytes) utilisés comme registres *memory mapped* afin d'interfacer les I/O intégrés. La *memory map* définit la correspondance entre la mémoire intégrée et une mémoire externe potentielle, et l'espace d'adressage de 16 bits. Elle est définie par le mode du microcontrôleur.

[1] Les schémas explicatifs des différents modes d'adressage sont donnés en annexe.

[2] Une mask ROM est une mémoire non volatile dont le contenu est fixée lors de la fabrication. Son contenu ne peut donc être modifié par l'utilisateur (au contraire des EEPROM par exemple).

Modes Opératoires Les modes opératoires du microcontrôleur sont définis par les niveaux de 2 pattes d'entrée appelées MD0 et MD1. Ces 2 bits peuvent être accédés comme bit 0 et bit 1 du *Mode Control Register* (MDCR) à l'adresse 0xFFC5. Il existe donc 4 modes différents :

1. Mode 0 : inopérant dans la famille des H8/3297.
2. Mode 1 (*expanded mode*) : adressage augmenté, ROM intégrée activée.
3. Mode 2 (*expanded mode*) : adressage augmenté, ROM intégrée désactivée.
4. Mode 3 (*single-chip mode*) : adressage limité, ROM intégrée activée.

Dans tous les modes, la RAM et le champ de registres intégrés sont accessibles aux mêmes adresses mémoires. La figure A.4 représente l'espace d'adressage dans les différents modes opératoires.

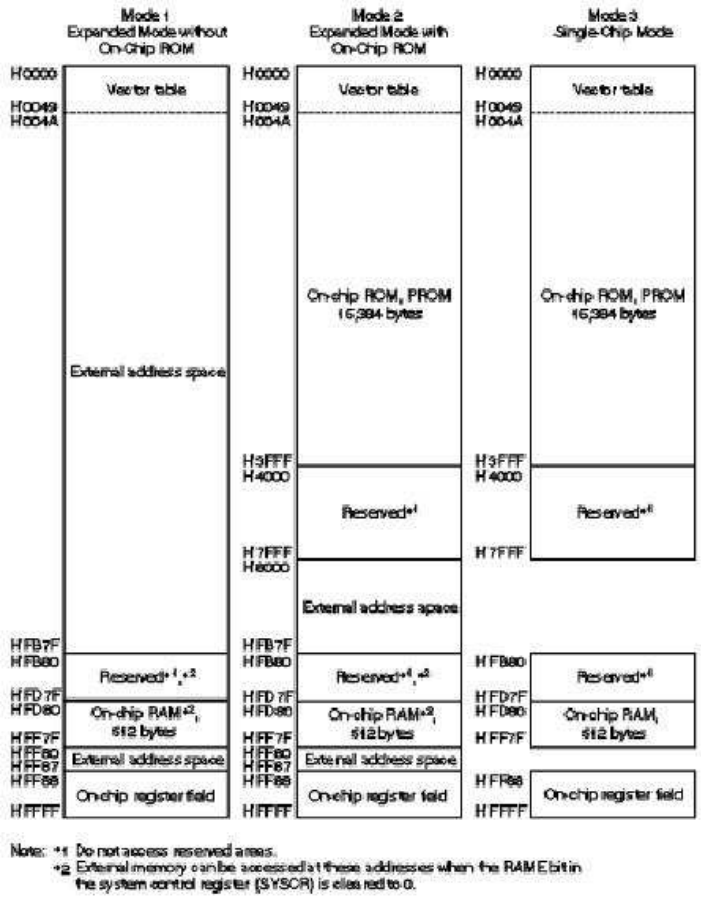


FIG. A.4 – Espace d'adressage H8/3292 (figure tirée du manuel [47]).

A.1.1.3 Les Entrées/Sorties Intégrées

Il en existe plusieurs dont certaines facilitent la connexion de périphériques.
- Timers : trois types de timers sont présents dans le H8/3292. Un timer 16-bit évoluant de manière libre, un timer 8-bit double canal[3] ainsi qu'un timer *watchdog*.
- Une interface pour communication sérielle.

[3]Il existe deux registres pour les constantes de temps. deux timers, s'incrémentant simultanément, peuvent être logiquement utilisés.

- Un convertisseur analogique/digital 10-bit.
- Ports de sorties.
- Ports d'entrées.
- Un contrôleur d'interruptions.

Les timers peuvent être utilisés sans circuit ajouté.

L'interface de communication sérielle est connectée au périphérique de communication infrarouge inclus dans le RCX.

Le convertisseur A/D possède une résolution de 10 bits, 8 canaux d'entrée (par le port 7 du H8/3292), ainsi que deux modes de conversion : le *single mode* qui permet de convertir un canal spécifique et le *scan mode* qui convertit un à quatre canaux en continu. Quatre registres 16-bit lui sont dédiés et une interruption de fin de conversion A/D (ADI) peut être demandée.

Les ports de sorties sont utilisés à travers des registres *device*. Lorsque le processeur met à jour un de ces registres, la nouvelle valeur est écrite sur la ligne ouput correspondante. Par exemple, une *LED* peut y être directement connectée.

Les ports d'entrées fonctionnent de manière analogue aux ports de sorties. Par exemple, un bouton peut y être directement connecté.

Le contrôleur d'interruptions fournit un mécanisme de gestion d'interruptions pour les interruptions internes (au nombre de 19) et externes (au nombre de 4). Son fonctionnement sera abordé plus en détail par la suite.

A.2 La Mémoire du RCX

Le mode choisi pour le microcontrôleur du RCX est le mode 2 ce qui permet d'utiliser à la fois la ROM et la RAM internes, ainsi que la RAM externe. L'ensemble de la mémoire du RCX est divisée en quatre parties décrites ci-dessous. Le choix du mode et de la connexion de 32 Kbytes de RAM externe nous permet de dresser une carte de la mémoire du RCX (voir table A.1).

Intervalle d'adresses	Type de mémoire	Contenu
0x0000 -> 0x0049	Mask ROM intégrée	Table des vecteurs d'interruptions microcontrôleur
0x004A -> 0x3FFF	Mask ROM intégrée	RCX *executive*
0x8000 -> 0xFB7F	RAM externe	Programme/données
0xFD80 -> 0xFF7F	RAM intégrée	Registres device RCX
0xFF80 -> 0xFF87	SRAM externe (~registres externes)	Table des vecteurs d'interruptions RCX/Programme/données
0xFF88 -> 0xFFFF	Registres intégrés	Registres device microcontrôleur

TAB. A.1 – Carte de la mémoire.

Type de stockage. Les données sont stockées de manière *little-endian* ce qui signifie que le byte le moins significatif est stocké à l'adresse plus petite. Toutes les zones mémoires sont accédées par des adresses 16-bit.

A.2.1 Les registres

A.2.1.1 Les registres à usages généraux (processeur)

Ils sont constitués de 8 registres de 16 bits. R0 à R6 peuvent configurés en 8 registres 16-bit ou en 16 registres 8-bit (par exemple : R1H, R1L, R3H, ...). R7 est le pointeur vers le dessus du stack (registre 16-bit obligatoirement, voir figure A.5). Ils sont directement accessibles par le programmeur et permettent d'utiliser les instructions registre à registre.

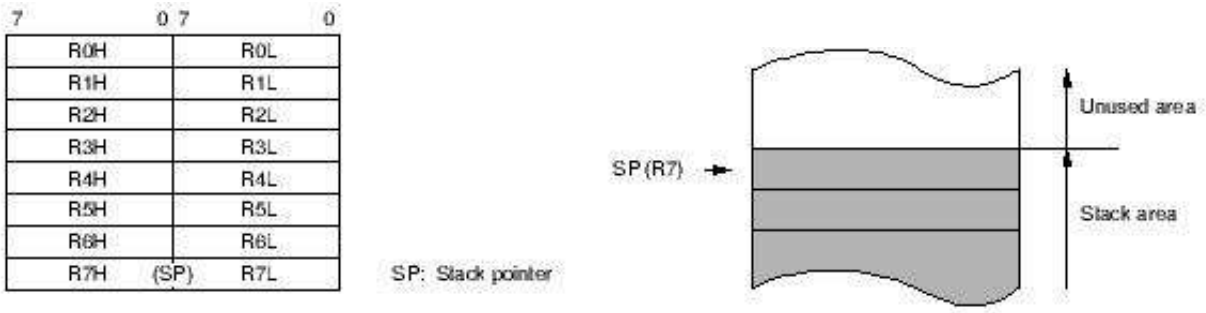


FIG. A.5 – Registres à usages généraux (figure tirée dedu manuel [47]).

A.2.1.2 Les registres de contrôle

Processeur

Il existe 2 registres de contrôle (voir figure A.6) : le *program counter* (PC) et le *condition code register* (CCR).

Le PC fournit l'adresse 16-bit de la prochaine instruction à exécuter. Etant donné que chaque instruction est sur 16 bits (1 mot), le bit le moins significatif du PC est toujours lu à 0.

Le CCR fournit toute une série d'informations concernant les instructions (carry flag, overflow flag, ...) et les interruptions (interrupt mask bit). Il est à remarquer que 2 bits du CCR sont réservés pour le programmeur.

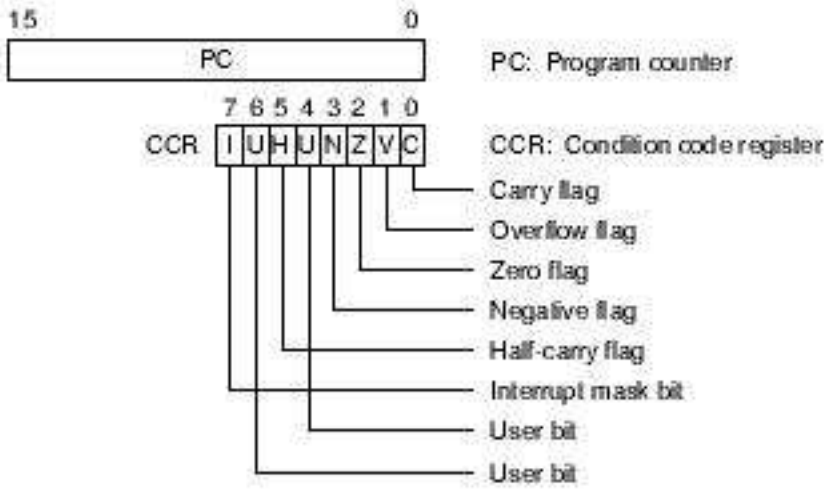


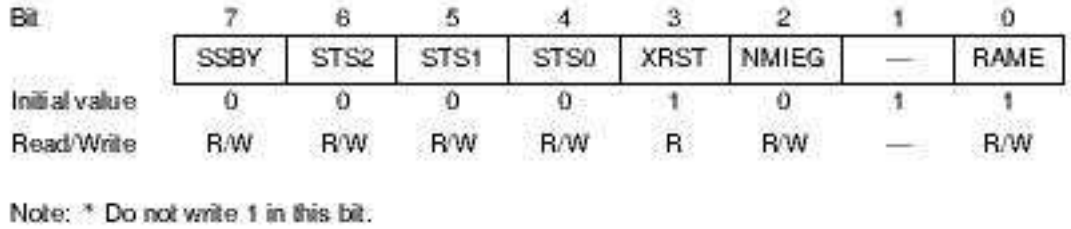
FIG. A.6 – Registre de contrôle (figure tirée du manuel [47]).

Microcontrôleur

Il existe plusieurs registres de contrôle au niveau du microcontrôleur : le *mode control register* (MDCR), le *system control register* (SYSCR) et les registres relatifs aux interruptions (ISCR, IER).

Le SYSCR est un registre 8-bit contrôlant l'opération de la puce (voir figure A.7).

On peut citer le *software standby* (SSBY) qui active la transition vers le standby software, l'*external reset* (XRST) qui spécifie la source de reset externe (watchdog ou patte d'entrée), ...

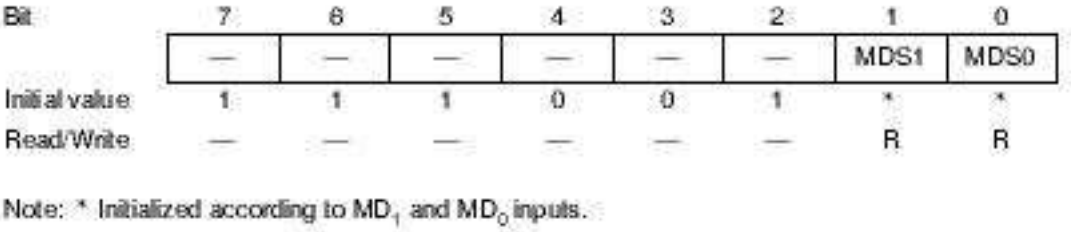


Bit	7	6	5	4	3	2	1	0
	SSBY	STS2	STS1	STS0	XRST	NMIEG	—	RAME
Initial value	0	0	0	0	1	0	1	1
Read/Write	R/W	R/W	R/W	R/W	R	R/W	—	R/W

Note: * Do not write 1 in this bit.

FIG. A.7 – SYSCR (figure tirée du manuel [47]).

Le MDCR indique le mode opératoire du microcontrôleur (voir figure A.8).



Bit	7	6	5	4	3	2	1	0
	—	—	—	—	—	—	MDS1	MDS0
Initial value	1	1	1	0	0	1	*	*
Read/Write	—	—	—	—	—	—	R	R

Note: * Initialized according to MD_1 and MD_0 inputs.

FIG. A.8 – MDCR (figure tirée du manuel [47]).

Les registres relatifs aux interruptions, à savoir le *IRQ sense sontrol register* (ISCR) et le *IRQ enable register* (IER), seront vus en détails dans un chapitre ultérieur sur le contrôleur d'interruptions.

A.2.1.3 Les registres device

Microcontrôleur

Il y a 128 bytes réservés sur la puce pour les registres device. Tout comme les périphériques externes, ils ne sont accessibles que par un bus de données 8-bit. Il faut donc deux cycles processeur pour lire un registre 16-bit.

RCX

Chaque périphérique peut être vu en deux parties : un contrôleur et le périphérique en lui-même. Prenons le cas des quatre boutons sur la face supérieure du RCX : vu de l'intérieur, le contrôleur est simplement 4 bits dans des registres device de deux ports[4] d'I/O du H8/3292 et qui représentent l'état des boutons (enfoncé ou libre). Vu de l'extérieur, ce sont de simples pièces de plastique.

Un programme tournant sur le RCX communique et synchronise les périphériques de la manière usuelle, c-à-d à travers les registres device et les interruptions ce qui constitue l'interface du contrôleur.

La plupart des contrôleurs des périphériques RCX sont basés sur les composants I/O du H8/3292 (voir table A.3). Il en résulte que ces contrôleurs utilisent le champ de registres intégrés et les interruptions fournies par les composants I/O du microcontrôleur. La seule exception se trouve être les trois ports de sorties du RCX qui sont gérés par un circuit électronique ajouté. Ainsi, ses registres device ne se trouvent pas en mémoire interne mais en RAM externe (et donc dans l'espace d'adressage externe, accessible par les pattes d'adresses/données du H8/3292).

Initialisation. Les registres ne sont pas initialisés au démarrage ni à la réinitialisation excepté le PC et l'interrupt mask bit dans le CCR. La valeur du PC est chargé de la table des vecteurs du contrôleur (adresse du RCX executive) et l'interrupt mask bit du CCR est mis à 1. Les autres registres doivent être initialisés par software. C'est notamment le cas du pointeur de stack qui doit être initialisé lors de la première instruction exécutée. Cette séquence d'opérations sera vue en détail lors de l'étude du *reset*.

[4]Chaque port possède un *data direction register* (DDR) et un *data register* (DR). Le DDR permet de spécifier si c'est un port d'entrée ou de sortie. Le DR permet de stocker les données du port.

Périphérique RCX contrôlé	Registres device	Interruptions
Boutons	Registres device du port I/O 4 et 7 : 2 lignes d'entrée du port 4 et du port 7 utilisées.	Run est connecté à l'IRQ0 et OnOff sur l'IRQ1
Ports d'entrée	Registres device du convertisseur A/D et du port I/O 6. 3 lignes de sortie du port 6 utilisées	ADI
Niveau du voltage batterie	Registres device du convertisseur A/D	ADI
Timers	Registres device des timers	Interruptions timer
Emetteur/transmetteur infra-rouge	Registres device du SCI, port I/O 4 et timer 1. Une ligne de sortie du port 4 utilisée	Interruption SCI et timer 1
LCD	Registres device du port I/O 6. 2 lignes de sortie utilisées	Pas d'interruption
Haut-parleur	Registres device du port I/O 6 et timer 0. 1 ligne de sortie utilisée	Interruption timer 0
Ports de sortie	Registres device dans l'espace d'adressage de la RAM externe	Pas d'interruption

TAB. A.3 – Périphériques internes.

A.2.2 La RAM Interne

Le contrôleur H8/3292 du RCX possède 512 bytes de RAM intégrée. Son activation est contrôlée par le bit *random access memory enable* (RAME) dans le *system control register* (SYSCR). Si le bit RAME est à 0, la RAM intégrée est activée sinon désactivée. Dans le cas du RCX, cette RAM est activée et est accessible aux adresses 0xFD80 -> 0xFF7F.

Elle est connectée au processeur par un bus de données 16-bit ainsi qu'un bus d'adressage 16-bit. Un byte ou un mot peuvent donc être accédés en deux états[5] processeur ce qui permet des extractions des données et des exécutions d'instruction rapides. La figure A.9 illustre le schéma bloc de la RAM intégrée.

A.2.3 La ROM interne

Les remarques concernant cette mémoire sont similaires à la RAM interne. La différence se situe donc au niveau de la persistance des données ainsi qu'à l'accès en lecture seule. Dans le cas du RCX, cette ROM de 16 Kbytes est activée et accessible aux adresses 0x0000 -> 0x3FFF. La figure A.10 illustre le schéma bloc de la ROM intégrée. Le RCX executive est contenu dans cette ROM interne.

A.2.4 La SRAM externe

Aucune documentation n'est disponible quant au type de la SRAM externe du RCX et à sa connexion au microcontrôleur. Seule sa taille de 32 Kbytes et le bus de données 8-bit qui y accède sont connus. Le microcontrôleur donne la possibilité d'insérer des *wait states* lors de l'accès à cette mémoire grâce à un contrôleur de *wait states*.

[5]Un état est la période entre deux flancs montants du signal d'horloge. A ne pas confondre avec les cycles du processeur.

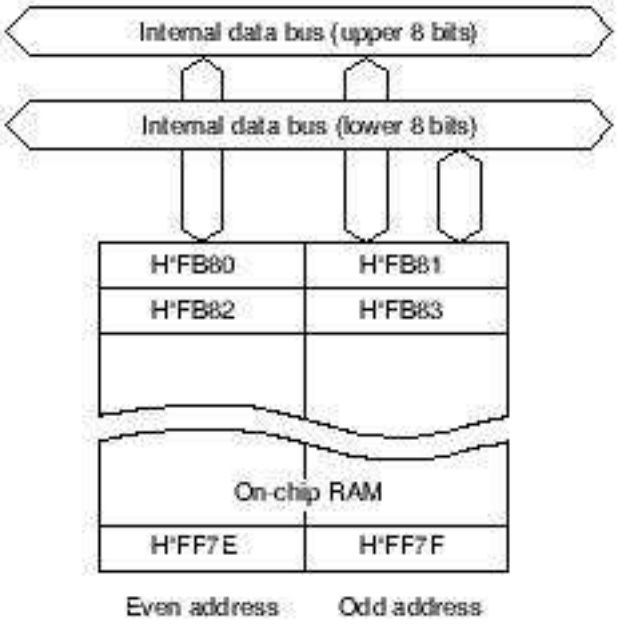


FIG. A.9 – Schéma bloc de la RAM intégrée (figure tirée du manuel [47]).

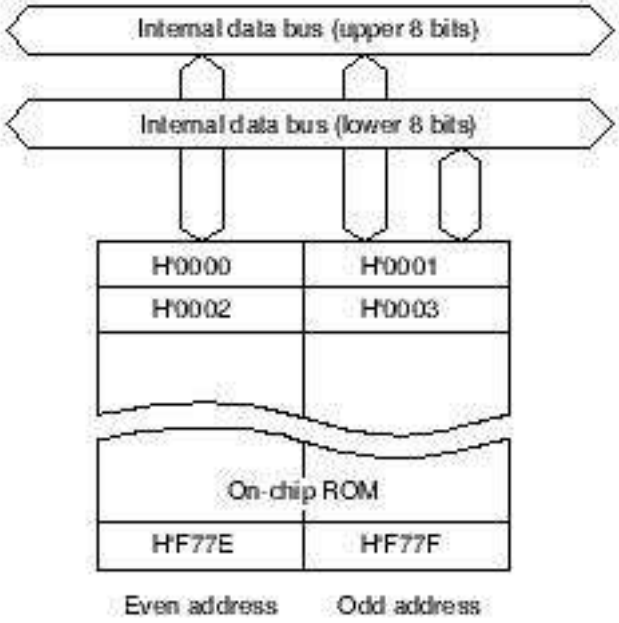


FIG. A.10 – Schéma bloc de la ROM intégrée (figure tirée du manuel [47]).

A.3 Le Contrôleur d'Interruptions

A.3.1 Vue d'ensemble

Aussi appelé *priority interrupt controller* (PIC)[6].

La famille des H8/3297 reconnaissent 2 types d'exceptions : les interruptions et le reset (voir table A.4).

Priorité	Type d'exception	Instant de détection	Instant du traitement de l'exception
Haute	Reset	Synchronisé avec l'horloge	La séquence du traitement hardware de l'exception commence dès que le niveau d'entrée de la patte RES change de bas à haut
Basse	Interruption	Fin de l'exécution de l'instruction	Lorsqu'une interruption est demandée, le traitement hardware de l'exception commence à la fin de l'instruction courante, ou à la fin de la séquence de traitement d'exception en cours

TAB. A.4 – Types d'exceptions.

A.3.2 Reset

Le reset a la plus haute priorité. Lorsque le RES passe à l'état bas ou lorsqu'il y a un reset du timer *watchdog* (par l'activation de l'option reset pour un *overflow*), tout processus courant stoppe et le microcontrôleur passe en état *reset*. L'état interne du processeur et les registres internes sont initialisés. La séquence de traitement commence dès que le RES passe de l'état bas à l'état haut, ou à la fin de l'impulsion du timer watchdog.

A.3.2.1 Séquence du Reset

Voici les étapes principales de la séquence effectuée lors d'une exception de type reset :

1. L'état interne du processeur et les registres internes sont initialisés. En outre, le I bit dans le CCR est mis à 1 afin d'ignorer toute nouvelle interruption.
2. Le processeur charge le PC à partir du premier mot de la table de vecteurs (stocké à l'adresse 0X0000 et 0X0001) et démarre ensuite l'exécution du programme. Dans le cas du RCX, ce programme est le RCX executive.

A.3.2.2 Désactivation des Interruptions Après Reset

Après un reset, si une interruption venait à être acceptée avant l'initialisation du pointeur de stack, le PC et le CCR pourraient ne pas être sauvés convenablement ce qui conduirait au plantage du programme. Afin d'éviter ce cas, toutes les interruptions, même la *non maskable interruption* (NMI), sont désactivées immédiatement après le reset. Ainsi, la première instruction du programme est toujours exécutée. Celle-ci devrait initialiser le pointeur de stack mais le registre CCR peut d'abord être initialisé.

[6] A ne pas confondre avec le MicroChip.

A.3.3 Interruptions

Il existe plusieurs sources d'interruptions incluant 4 interruptions externes (NMI, IRQ0 -> IRQ2) et 19 interruptions internes. Pour chaque interruption sont définies une priorité et une adresse de vecteur. Les caractéristiques de ces interruptions sont :

- NMI a la plus haute priorité au sein des interruptions et est toujours acceptée (sauf au cours d'un reset). Toutes les autres interruptions externes et internes peuvent être masquées par le bit I dans le CCR.
- IRQ0 -> IRQ2 peuvent être détectées soit par un flanc descendant soit par le niveau du signal d'entrée.
- Toutes les interruptions ont leur propre adresse de vecteur. Ainsi, une routine de traitement software d'interruption ne devra pas vérifier le type d'interruption à traiter.
- Le timer *watchdog* peut générer soit un NMI, soit une interruption *overflow* selon les effets recherchés.

A.3.3.1 Registres Concernant les Interruptions

Il y a 3 registres concernés : SYSCR, ISCR et IER.

Le SYSCR, déjà mentionné auparavant comme registre de contrôle, contient le bit *nonmaskable interrupt edge* (NMIEG) qui détermine le type de génération d'une interruption non masquable : 0 = flanc descendant (état initial), 1 = flanc montant.

Le *IRQ sense control register* (ISCR) permet de spécifier si chaque IRQ est déclenché lors d'un flanc descendant ou du maintien d'un niveau (voir figure A.11).

Bit	7	6	5	4	3	2	1	0
	—	—	—	—	—	IRQ2SC	IRQ1SC	IRQ0SC
Initial value	1	1	1	1	1	0	0	0
Read/Write	—	—	—	—	—	R/W	R/W	R/W

FIG. A.11 – ISCR (figure tirée du manuel [47]).

Le *IRQ enable register* (IER) permet d'activer ou désactiver les 3 IRQ individuellement (voir figure A.12).

Bit	7	6	5	4	3	2	1	0
	—	—	—	—	—	IRQ2E	IRQ1E	IRQ0E
Initial value	1	1	1	1	1	0	0	0
Read/Write	—	—	—	—	—	R/W	R/W	R/W

FIG. A.12 – IER (figure tirée du manuel [47]).

A.3.3.2 Interruptions Externes

Ces interruptions sont NMI et IRQ0 -> IRQ2. Elles peuvent toutes quatre être utilisées pour sortir du mode *standby software* (voir états du processeur).

Lorsqu'une de ces interruptions est acceptée (I bit à 0 dans le CCR, IRQnE bit à 1 dans le IER), le I bit du CCR est mis à 1 afin d'éviter d'autres interruptions lors du traitement.

Chacune possède une adresse de vecteur d'interruption unique :

– NMI a le numéro 3 des vecteurs d'interruptions.

– IRQ0, IRQ1 et IRQ2 ont respectivement les numéros 4, 5 et 6.

En outre, les priorités parmi les IRQ sont les suivantes : IRQ2 < IRQ1 < IRQ0. Les interruptions IRQ ne dépendent pas du type de patte, que ce soit une patte de sortie ou d'entrée sur le port 4. En effet, il suffit de configurer le DDR bit du port 4 sur 0 afin que ces pattes soit considérées comme pattes d'entrées par le microcontrôleur (attention, elles ne peuvent donc plus être utilisées comme sorties pour les timers, l'interface de communication sérielle ou pour le convertisseur A/D).

A.3.3.3 Interruptions Internes

Leurs caractéristiques ont déjà été reprises plus haut. On peut cependant ajouter que leurs numéros de vecteurs d'interruption s'échelonnent de 12 à 36.

A.3.3.4 Traitement des Interruptions

Les interruptions sont contrôlées par un contrôleur d'interruptions qui effectue un choix arbitraire entre des requêtes d'interruptions simultanées et commande le processeur afin de démarrer la séquence de traitement hardware de gestion d'exception. De plus, il fournit le numéro de vecteur nécessaire au processeur. Le schéma bloc de la figure A.13 illustre le contrôleur d'interruption.

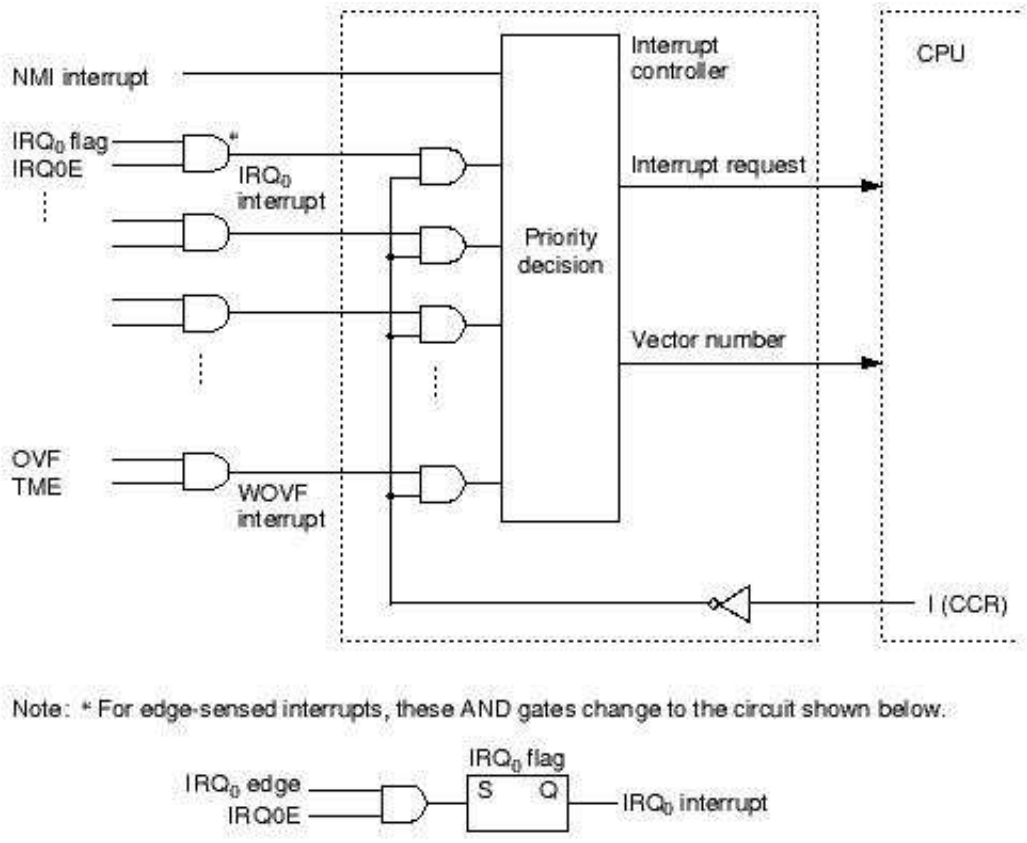


FIG. A.13 – Contrôleur d'interruption (figure tirée du manuel [47]).

Les interruptions IRQ et des modules intégrés (excepté le *reset* pour le timer *watchdog*) ont tous un bit d'activation correspondant. Si ce bit est à 0, la requête d'interruption est ignorée et n'est donc pas envoyée au contrôleur d'interruption. De même, toutes les interruptions peuvent être masquées si le bit I du CCR est à 1. Si les deux conditions précédentes ne sont pas remplies, la requête est envoyée au contrôleur d'interruption.

Le NMI est toujours accepté, excepté dans l'état reset ou standby hardware.

Une fois que le contrôleur gère une requête valide, il la transfère au processeur ainsi que son numéro de vecteur. Lorsque plus d'une interruption sont demandées simultanément, le contrôleur se base sur leur priorité pour fixer l'ordre de traitement de celles-ci par le processeur. Quand une requête d'interruption est notifiée au processeur, à la fin de l'instruction courante ou de la séquence de traitement d'un requête d'interruption précédente, le processeur commence la séquence de traitement et verrouille en mémoire le numéro de vecteur. L'organigramme de la figure A.14 illustre le traitement d'une interruption.

1. Une requête d'interruption est envoyée au contrôleur lorsque NMI se produit et lorsqu' un IRQ ou une interruption interne d'un des modules du H8/3292 dont le bit d'activation est à 1.
2. Le contrôleur d'interruption vérifie le bit I du CCR et accepte l'interruption si celui-ci est à 0. S'il est à 1, seule la requête NMI est acceptée, les autres restent en attente.
3. Parmi toutes les requêtes acceptées, le contrôleur d'interruption sélectionne la requête de priorité la plus élevée et la transfère au processeur. Les autres requêtes restent en attente.
4. Lorsqu'il reçoit une requête d'interruption, le processeur attend l'accomplissement de l'instruction courante ou de la séquence de traitement hardware d'une exception en cours, et commence ensuite la prochaine séquence de traitement en verrouillant en mémoire le numéro de vecteur.
5. Dans la séquence de traitement hardware de l'exception, le processeur sauvegarde d'abord sur le stack le PC et le CCR. Le PC sur le stack représente l'adresse de la première instruction qui sera exécutée au retour de la routine software du traitement de l'exception.
6. Ensuite, le bit I du CCR est mis à 1 afin de masquer toutes les interruptions ultérieures excepté le NMI.
7. L'adresse du vecteur correspondante au numéro de vecteur est générée, l'entrée dans la table des vecteurs à cette adresse est chargée dans le PC et l'exécution de la routine software de traitement de l'exception est lancée à l'adresse de cette entrée.

Stack Voyons un peu plus en détails le fonctionnement du stack (voir figure A.15).

Le stack est géré au niveau du mot (2 bytes). Durant le processus d'interruption, le CCR est sauvé sur le stack mais la taille de celui-ci ne fait que 8 bits. il est donc sauvé en tant qu'un mot en dupliquant ces données (2x8 bits). Lors du retour d'une routine d'exception par l'exécution d'une instruction de contrôle du système, *return from exception-handling routine* (RTE), le PC et le CCR sont restaurés. dans le cas du CCR, le second byte redondant à l'adresse impaire est ignoré.

Note sur la gestion du stack. Lors d'un accès de type mot (comme pour le stack), le bit le moins significatif de l'adresse est toujours considéré comme valant 0. Il faut donc prendre garde à garder une adresse paire dans le pointeur de stack. Voici un exemple utilisant les instruction PUSH et POP (ou MOV.W Rn,@-SP et MOV.W @SP+,Rn) pour sauvegarder et retirer des valeurs du stack.

Mettre une adresse impaire dans le pointeur de stack peut entraîner un crash du programme (perte d'une partie du PC, voir figure A.16). On remarque donc que la partie haute du PC est perdue ce qui entraînera une mauvaise valeur du PC lors de son extraction et donc un crash du programme.

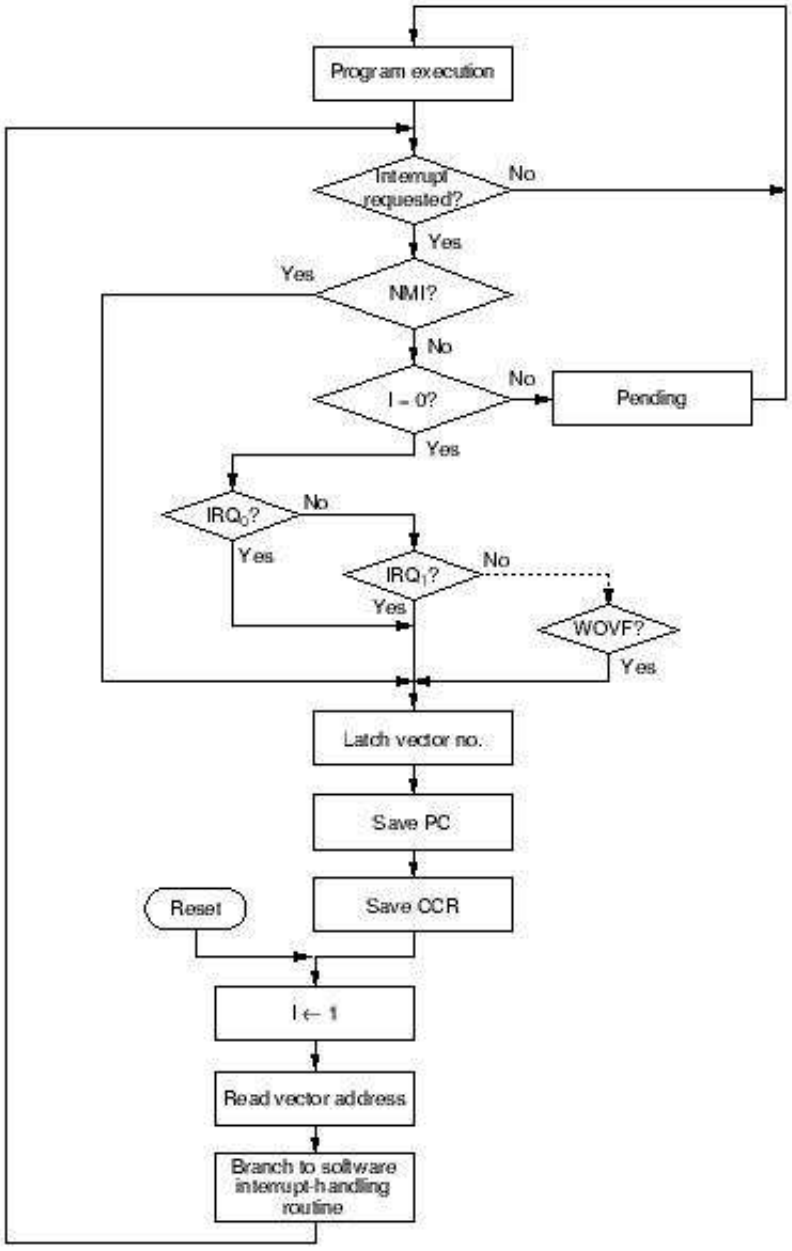
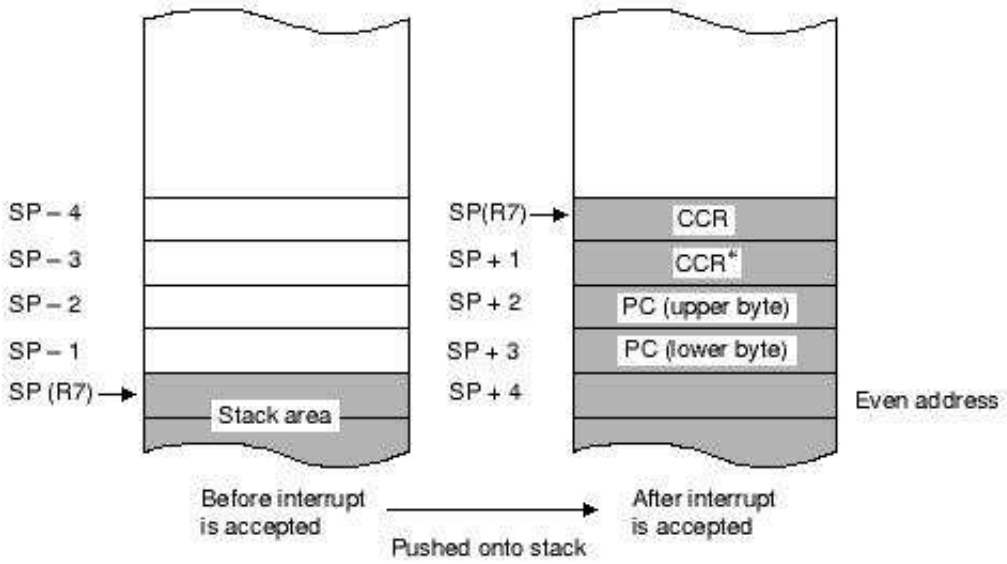


FIG. A.14 – Organigramme du contrôleur d'interruption (figure tirée du manuel [47]).



PC: Program counter
CCR: Condition code register
SP: Stack pointer

Notes: 1. The PC contains the address of the first instruction executed after return.
2. Registers must be saved and restored by word access at an even address.

* Ignored on return.

FIG. A.15 – Usage du stack lors d'une interruption (figure tirée du manuel [47]).

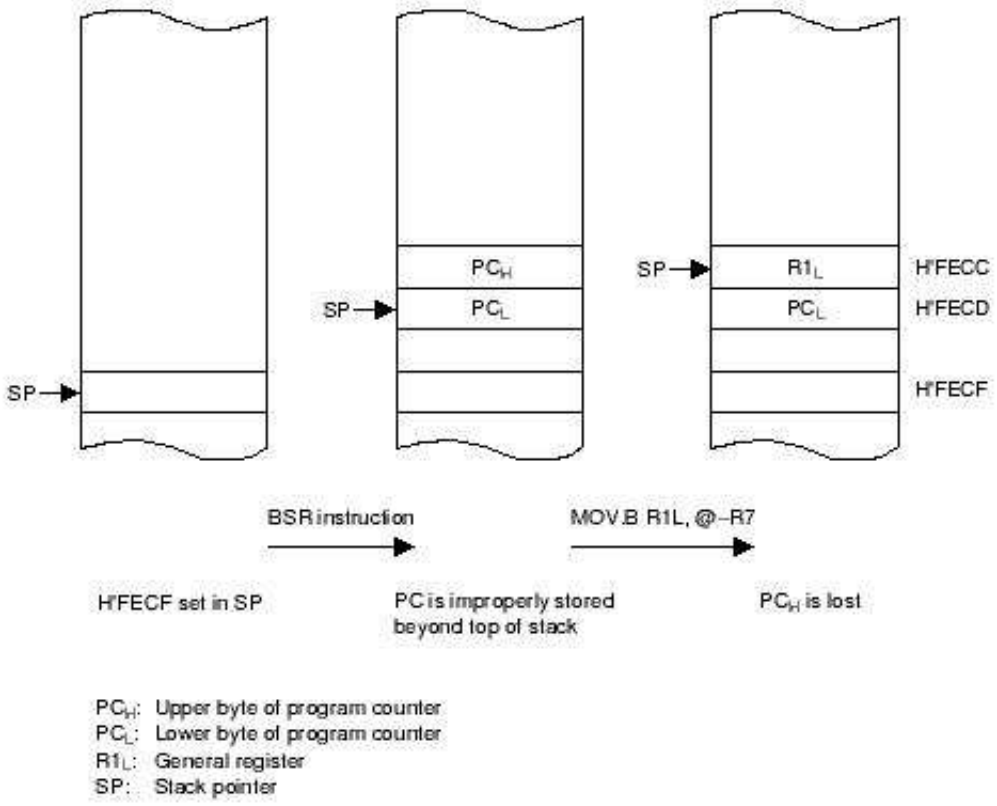


FIG. A.16 – Dommage causé par une adresse impaire dans SP (figure tirée du manuel [47]).

Annexe B

Analyse Statistique du Senseur de Luminosité

B.1 Code Source

comparison_sensors.m

```
function [] = comparison_sensors(legomat1, legomat2, legomat3, nb)
%COMPARISON_SENSORS Summary of this function goes here
%  Detailed explanation goes here
data_nop = load(legomat1,'V1', 'V2', '-mat') ;
data_mean = load(legomat2,'V1', 'V2', '-mat') ;
data_median = load(legomat3,'V1', 'V2', '-mat') ;
if nb <= 0
    nb = length(data_nop.V1) ;
end
if nb > length(data_nop.V1)
    nb = length(data_nop.V1) ;
end
if nb > length(data_mean.V1)
    nb = length(data_mean.V1) ;
end
if nb > length(data_median.V1)
    nb = length(data_median.V1) ;
end
fprintf('Technique NOP :\n\nsenseur gauche :\nmean : %g\nmedian :
%g\nvariance : %g\namplitude : %g\n\n', ...
    mean(data_nop.V1(1 :nb)), median(data_nop.V1(1 :nb)), var(data_nop.V1(1 :nb)),
max(data_nop.V1(1 :nb))-min(data_nop.V1(1 :nb))) ;
fprintf('senseur droit :\nmoyenne : %g\nmediane : %g\nvariance :
%g\namplitude : %g\n\n\n', ...
    mean(data_nop.V2(1 :nb)), median(data_nop.V2(1 :nb)),
var(data_nop.V2(1 :nb)),
max(data_nop.V2(1 :nb))-min(data_nop.V2(1 :nb))) ;
fprintf('Technique MEAN :\n\nsenseur gauche :\nmoyenne : %g\nmediane :
%g\nvariance : %g\namplitude : %g\n\n', ...
    mean(data_mean.V1(1 :nb)), median(data_mean.V1(1 :nb)), var(data_mean.V1(1 :nb)),
max(data_mean.V1(1 :nb))-min(data_mean.V1(1 :nb))) ;
fprintf('senseur droit :\nmoyenne : %g\nmediane : %g\nvariance :
%g\namplitude : %g\n\n\n', ...
    mean(data_mean.V2(1 :nb)), median(data_mean.V2(1 :nb)),
var(data_mean.V2(1 :nb)),
```

```
max(data_mean.V2(1 :nb))-min(data_mean.V2(1 :nb))) ;
fprintf('Technique MEDIAN :\n\nsenseur gauche :\nmoyenne : %g\nmediane :
%g\nvariance : %g\namplitude : %g\n\n', ...
    mean(data_median.V1(1 :nb)), median(data_median.V1(1 :nb)),
var(data_median.V1(1 :nb)),
max(data_median.V1(1 :nb))-min(data_median.V1(1 :nb))) ;
fprintf('senseur droit :\nmoyenne : %g\nmediane : %g\nvariance :
%g\namplitude : %g\n\n\n', ...
    mean(data_median.V2(1 :nb)), median(data_median.V2(1 :nb)),
var(data_median.V2(1 :nb)), max(data_median.V2(1 :nb))-min(data_median.V2(1 :nb))) ;
subplot(1,2,1) ;
[f, x, u] = ksdensity(data_nop.V1(1 :nb)) ;
[f, x] = ksdensity(data_nop.V1(1 :nb), 'width', u/2) ;
plot(x, f, 'r') ;
title('Distribution de probabilite du senseur de lumiere gauche') ;
xlabel('senseur gauche') ;
ylabel('proba') ;
hold on
[f, x, u] = ksdensity(data_mean.V1(1 :nb)) ;
[f, x] = ksdensity(data_mean.V1(1 :nb), 'width', u/2) ;
plot(x, f, 'b') ;
[f, x, u] = ksdensity(data_median.V1(1 :nb)) ;
[f, x] = ksdensity(data_median.V1(1 :nb), 'width', u/2) ;
plot(x, f, 'g') ;
hold off
legend('Technique NOP', 'Technique MEAN', 'Technique MEDIAN', 1) ;
subplot(1,2,2) ;
[f, x, u] = ksdensity(data_nop.V2(1 :nb)) ;
[f, x] = ksdensity(data_nop.V2(1 :nb), 'width', u/2) ;
plot(x, f, 'r') ;
title('Distribution de probabilite du senseur de lumiere droit') ;
xlabel('senseur droit') ;
ylabel('proba') ;
hold on
[f, x, u] = ksdensity(data_mean.V2(1 :nb)) ;
[f, x] = ksdensity(data_mean.V2(1 :nb), 'width', u/2) ;
plot(x, f, 'b') ;
[f, x, u] = ksdensity(data_median.V1(1 :nb)) ;
[f, x] = ksdensity(data_median.V2(1 :nb), 'width', u/2) ;
plot(x, f, 'g') ;
hold off
legend('Technique NOP', 'technique MEAN', 'Technique MEDIAN', 1) ;
```

Annexe C

Contrôle des Moteurs

C.1 Pseudocode

Algorithme 6 Pseudocode de l'algorithme de contrôle des moteurs grâce aux senseurs de rotation. Les données d'entrée sont des entiers compris de -255 à 255.

```
INPUT : nombre de rotations à gauche et nombre de rotations à droite

Transformer le nombre de rotations à gauche en nombre de ticks du senseur de rotation gauche
Transformer le nombre de rotations à droite en nombre de ticks du senseur de rotation droit
ticks=min(ticks_gauche, ticks_droit)
WHILE ticks!=0
  //Les deux roues doivent tourner en même temps avec la meilleure synchronisation possible
  Actionner les deux moteurs à vitesse rapide dans le sens spécifié par les données d'entrée
  Vérifier le compteur du senseur de rotation gauche et droit
  IF retard important entre compteur de rotation gauche et droit THEN
    //un moteur est significativement moins rapide que l'autre
    Actionner le moteur trop rapide à une vitesse lente
  ELSE
    IF retard faible entre compteur de rotation gauche et droit THEN
      //Un moteur est légèrement moins rapide que l'autre
      Actionner le moteur trop rapide à une vitesse moyenne
    ELSE
      //Les deux moteurs sont synchronisés
      Actionner les deux moteurs à une vitesse rapide
    ENDIF
  ENDIF
  Calculer le nombre de ticks restants
ENDWHILE
ticks=|ticks_gauche − ticks_droit|
WHILE ticks!=0
  //Une seule roue doit tourner pour le nombre de ticks restants
  Actionner le moteur de la roue devant tourner à une vitesse rapide dans le sens spécifié
    par les données d'entrée
  Vérifier le compteur du senseur de rotation du côté de la roue considérée
  Calculer le nombre de ticks restant
ENDWHILE
```

C.2 Code Source

servomotor.h

```
/*
  low level control library : 2 rotation sensors and 2 motors are needed
*/
#ifndef __servomotor_h__
#define __servomotor_h__
#include <dmotor.h>
#include <dsensor.h>
#include <unistd.h>
#include <conio.h>
#define TICKS_PER_LITTLE_ROT 20//check every 20 ticks for sync between the two wheels
#define ROT_RATIO 40 //for the function drive_motors
#define STALL_SAMPLES 5 //how many samples should we take before calling a stall ?
#define STALL_INTERVAL 100//time interval between each sample point in ms
#define MAXMOTOR 2//max number of motor
#define MAX_SPEED_MOTOR 255
#define MIN_SPEED_MOTOR 0
#define HIGH_MOTOR_SPEED 255//255 is the real max
#define MEDIUM_MOTOR_SPEED 200
#define LOW_MOTOR_SPEED 180
#define SPEED_INTERVAL 45
typedef struct motor_s {
  unsigned char stalled ;
  MotorDirection dir ;
  volatile int* rot_sensor ;//to take measures
  volatile int* rot_sensor_port ;//for initialisation
} motor_servo ;
motor_servo motors[MAXMOTOR] ;
#define MOTOR_L motors[0]
#define MOTOR_R motors[1]
//functions
//apply a rotation of angl to the wheel on motor m
int drive_motors(int rotleft, int rotright) ;
int init_motor(motor_servo* m, int* rs, int* rsp) ;
void init_robot() ;
int stall_thread(int argc, char** argv) ;
int stall_control() ;
#endif //__servomotor_h__
```

servomotor.c

```
int drive_motors(int rotleft, int rotright) {
  //this low level function make turn the right wheel of rotleft/ROT_RATIO
  //and the left wheel of rotright/ROT_RATIO
  //algo : go synch for the common part between rotleft and rotright
  //after, only one motor is in action for the number of rot remaining
  volatile int* sensorr ;
  volatile int*  sensorl ;
  int arotleft = 0, arotright = 0, nbr_rot_todo = 0, val1r = 0, val1l = 0, val2r = 0, val2l = 0, nbr_rotl
  unsigned char msr = 0, msl = 0, booll = 0, boolr = 0 ;
  motor_servo* ml = &MOTOR_L ;
  motor_servo* mr = &MOTOR_R ;
  if (rotleft == 0 && rotright == 0)//nothing to do
    return 0 ;

  sensorl = ml->rot_sensor ;//to avoid dereferencing
```

```
sensorr = mr->rot_sensor ;//idem

//compute the common part
arotleft = my_abs(rotleft) ;
arotright = my_abs(rotright) ;
if (arotleft > arotright) {
  //real formula -> ticks_rest = (arotright / ROT_RATIO) * MOTOR_WHEEL_RATIO * 16 ;
  ticks_rest = arotright * 2 ;//common ticks to do
  arotleft = arotleft - arotright ;
  arotright = 0 ;
}
else {//rotleft <= rotright
  //real formula -> ticks_rest = (arotleft / ROT_RATIO) * MOTOR_WHEEL_RATIO * 16 ;
  ticks_rest = arotleft * 2 ;//common ticks to do
  arotright = arotright - arotleft ;//0 if equal
  arotleft = 0 ;
}
//keep the information of direction
if (rotleft < 0)
  rotleft = -1 ;
else
  rotleft = 1 ;
if (rotright < 0)
  rotright = -1 ;
else
  rotright = 1 ;
//first, the robot goes in synchronization during ticks_rest rotations
msr = HIGH_MOTOR_SPEED ;//initial value for motor
msl = HIGH_MOTOR_SPEED ;//idem
ticks_current = TICKS_PER_LITTLE_ROT ;//initial value for ticks

while (ticks_rest > 0) {//if ticks_rest == 0, the movement is over

  //how many ticks needed ? ticks_rest and ticks_current are always positive
  if (ticks_rest > ticks_current)
    nbr_rot_todo = ticks_current ;
  else
    nbr_rot_todo = ticks_rest ;
  //apply number of ticks to do on each motor
  nbr_rotr = nbr_rot_todo ;
  nbr_rotl = nbr_rot_todo ;
  if (boolr == 1)//changement of speed
    boolr = 0 ;
  else//inverse changement
    msr = HIGH_MOTOR_SPEED ;
  if (booll == 1)//changement of speed
    booll = 0 ;
  else//inverse changement
    msl = HIGH_MOTOR_SPEED ;
  if (rotright == 1) {//fwd in motor right
    val1r = *sensorr ;
    //update motor directions
    mr->dir = fwd ;
    motor_a_speed(msr) ;
    motor_a_dir(fwd) ;
  }
  else {//rev on motor right
    val1r = - *sensorr ;
```

```
  //update motor directions
  mr->dir = rev ;
  motor_a_speed(msr) ;
  motor_a_dir(rev) ;
}
if (rotleft == 1) {//fwd on motor left
  val1l = *sensorl ;
  //update motor directions
  ml->dir = fwd ;
  motor_c_speed(msl) ;
  motor_c_dir(fwd) ;
}
else {//rev on motor left
  val1l = - *sensorl ;
  //update motor directions
  ml->dir = rev ;
  motor_c_speed(msl) ;
  motor_c_dir(rev) ;
}
//at this point nbr_rot* are always positive
while((nbr_rotr > 0) || (nbr_rotl > 0)) {

  //verif if one motor is stalled
  if (mr->stalled == 1 || ml->stalled == 1) {
    stall_control() ;
    return -1 ;
  }
  msleep(SPEED_INTERVAL) ;
  //second valors of sensors
  if (rotright == 1)//fwd
    val2r = *sensorr ;
  else//rev
    val2r = - *sensorr ;
  if (rotleft == 1)//fwd
    val2l = *sensorl ;
  else//rev
    val2l = - *sensorl ;
  if(nbr_rotr > 0) {
    nbr_rotr -= (val2r-val1r) ;
    val1r = val2r ;
  }
  else {//no more rotation needed in this step for motor right
    //update motor direction

    if (nbr_rotl >= 2) {//great difference of speed
      //change the speed for one little round
      msr = LOW_MOTOR_SPEED ;
      boolr = 1 ;
    }
    else {//little speed diff
      //change the speed for one little round
      msr = MEDIUM_MOTOR_SPEED ;
      boolr = 1 ;
    }
  }

  if(nbr_rotl > 0) {
    nbr_rotl -= (val2l-val1l) ;
```

```
      val1l = val2l ;
    }
    else {//no more rotation needed in this step for motor left
      //update motor direction

      if (nbr_rotr >= 2) {//great difference of speed
        //change the speed for one little round
        msl = LOW_MOTOR_SPEED ;
        booll = 1 ;
      }
      else {//little speed diff
        //change the speed for one little round
        msl = MEDIUM_MOTOR_SPEED ;
        booll = 1 ;
      }
    }
  }
  //set motors to float for an intermediate movement
  mr->dir = off ;
  ml->dir = off ;
  motor_a_dir(off) ;
  motor_c_dir(off) ;
  ticks_rest -= nbr_rot_todo ;
}

//do the rest of the movement with one motor
if (arotleft == 0) {//right wheel to turn
  //real formula -> ticks_rest = (arotright / ROT_RATIO) * MOTOR_WHEEL_RATIO * 16 ;
  ticks_rest = arotright * 2 ;
  if (rotright == -1) {
    mr->dir = rev ;
    motor_a_dir(rev) ;
  }
  else {//rotright = 1
    mr->dir = fwd ;
    motor_a_dir(fwd) ;
  }
  motor_a_speed(HIGH_MOTOR_SPEED) ;
}
else {//left wheel to turn
  sensorr = sensorl ;//reuse of existing variables
  mr = ml ;
  rotright = rotleft ;
  //real formula -> ticks_rest = (arotright / ROT_RATIO) * MOTOR_WHEEL_RATIO * 16 ;
  ticks_rest = arotleft * 2 ;

  if (rotleft == -1) {
    ml->dir = rev ;
    motor_c_dir(rev) ;
  }
  else {//rotleft = 1
    ml->dir = fwd ;
    motor_c_dir(fwd) ;
  }
  motor_c_speed(HIGH_MOTOR_SPEED) ;
}
//first valor
if (rotright == 1)//fwd
```

```
    val1r = *sensorr;
  else//rev
    val1r = - *sensorr;

  while (ticks_rest > 0) {

    //verif if one motor is stalled
    if (mr->stalled == 1) {
      stall_control();
      return -1;
    }
    msleep(SPEED_INTERVAL);

    //second valor
    if (rotright == 1)//fwd
      val2r = *sensorr;
    else//rev
      val2r = - *sensorr;
    ticks_rest -= (val2r-val1r);
    val1r = val2r;
  }
  //reset valor of mr which has been probably modified
  mr = &MOTOR_R;
  //set motors to brake at the end of movement
  mr->dir = brake;
  ml->dir = brake;
  motor_a_dir(brake);
  motor_c_dir(brake);

  //reset the rotation number to avoid rotation overflow
  ds_rotation_set(ml->rot_sensor, 0);
  ds_rotation_set(mr->rot_sensor, 0);
  return 0;
}
int my_abs(int a) {
  return (a<0)? -a : a;
}
int stall_thread(int argc, char **argv) {
  int valuer, valuel;
  int sampleCountr, sampleCountl;
  int sensorReadingr, sensorReadingl;
  volatile int* sensorr;
  volatile int* sensorl;
  sensorr = MOTOR_R.rot_sensor_port;//take an initial reading
  sensorl = MOTOR_L.rot_sensor_port;
  valuer = *sensorr;
  valuel = *sensorl;
  sampleCountr = 0;//no samples have been taken so far
  sampleCountl = 0;
  //we live forever, or at least until the main loop kills us :-)
  while(1) {
    sensorReadingr = *sensorr;//take a reading
    sensorReadingl = *sensorl;//take a reading
    msleep(STALL_INTERVAL);
    //if the motor is currently disabled then don't bother to process
    //stall data
    //verif for motor right
    if (MOTOR_R.dir != off && MOTOR_R.dir != brake) {
```

```
        if(valuer != sensorReadingr) {
          //change is good. It means we're moving, so update value and
          //reset sampleCount

          valuer = sensorReadingr ;
          sampleCountr = 0 ;
        } else {
          //ok, time to take samples. If this is the last sample of the
          //set, then we call it a stall

          if(sampleCountr == STALL_SAMPLES) {

            valuer = sensorReadingr ;
            sampleCountr = 0 ;
            MOTOR_R.stalled = 1 ;
          } else {
            //update the sampleCount
            sampleCountr++ ;
          }
        }
      }
      //verif for motor left
      if (MOTOR_L.dir != off && MOTOR_L.dir != brake) {
        if(valuel != sensorReadingl) {
          //change is good. It means we're moving, so update value and
          //reset sampleCount

          valuel = sensorReadingl ;
          sampleCountl = 0 ;
        } else {
          //ok, time to take samples. If this is the last sample of the
          //set, then we call it a stall

          if(sampleCountl == STALL_SAMPLES) {
            valuel = sensorReadingl ;
            sampleCountl = 0 ;
            MOTOR_L.stalled = 1 ;
          } else {
            //update the sampleCount
            sampleCountl++ ;
          }
        }
      }
    }
  }
  int stall_control() {
   //basic stall control : reset the necessary variables to continue in a coherent state
    cputs("stall") ;
    //set motor directions to brake
    motor_a_dir(brake) ;
    motor_c_dir(brake) ;
    //reset motor direction for each motor
    MOTOR_L.dir = brake ;
    MOTOR_R.dir = brake ;
    //reset satll flag for each motor
    MOTOR_L.stalled = 0 ;
    MOTOR_R.stalled = 0 ;
    return 0 ;
```

```
}
void init_robot() {

  //this thread verifies if the robot is stalled by watching the rotation counts
  execi(stall_thread,0,NULL,PRIO_NORMAL,DEFAULT_STACK_SIZE) ;
}
int init_motor(motor_servo* m, int* rs, int* rsp) {
 //init motor informations
  m->stalled = 0 ;
  m->dir = brake ;
  m->rot_sensor = rs ;
  m->rot_sensor_port = rsp ;
  //active the rotation sensor
  ds_active(rsp) ;
  //ok, start sensor processing
  ds_rotation_set(rsp, 0) ;
  ds_rotation_on(rsp) ;
  return 0 ;
}
```

Annexe D

Protocole de Communication

Le pseudocode et code source du protocole de communication sont répartis en 3 parties : la partie embarquée à bord du RCX, la partie de l'ordinateur client et la partie de l'ordinateur central.

D.1 RCX

D.1.1 Pseudocode

Algorithme 7 Pseudocode du logiciel de protocole de communication embarqué à bord du RCX.

```
TROPOCOLRCX
  Lancer la tâche COMMANAGER
  Lancer la tâche PINGSENDER
  WHILE tâche active
    Attendre une commande
    Exécuter la commande
    Mettre à jour le numéro de séquence
    IF données à envoyer THEN
      Envoyer les données
      Mettre à jour le numéro de séquence
      Initialiser le timer du ping
    ENDIF
  ENDWHILE
ENDTROPOCOLRCX

THREAD PINGSENDER
  WHILE tâche active
    IF timer du ping à 0 THEN
      Envoyer un ping
      Initialiser le timer du ping
    ENDIF
  ENDWHILE
ENDTHREAD

THREAD COMMANAGER
  WHILE tâche active
    Attendre des données
    Vérifier la longueur des données
    Vérifier le CRC des données
    Vérifier le numéro de RCX de destination
    Vérifier le numéro de commande
    IF commande valide THEN
      Mettre à jour commande à exécuter
      Envoyer ACK
      Initialiser le timer du ping
    ENDIF
  ENDWHILE
ENDTHREAD
```

D.1.2 Code Source

tropocol.h

```
/*
  functions for reliable robot communications
  Protocol and commands
*/
#ifndef __tropocol_h__
#define __tropocol_h__
```

```
//#define DEBUG_TROPOCOL
#define NBR_BYTES_COMM 6
#define RCX_ID 3
#define PING_EXAMPLE 96
#define COLLISION_TIME_COEF1 30
#define COLLISION_TIME_COEF2 33
#define COLLISION_TIME (RCX_ID * COLLISION_TIME_COEF1)
#define PING_INTERVAL 250
#define TIME_MOTOR 500
#define LIGHT_MEAN 5
#define LIGHT_MEDIAN 5
#include <lnp.h>
#include <lnp-logical.h>
#include <unistd.h>
#include <dsensor.h>
#include <dsound.h>
#include <dmotor.h>
#include <conio.h>
#include <time.h>
#include "servomotor.h"
#include "lightmux.h"
#include <conio.h>
#ifdef DEBUG_TROPOCOL
#include <rom/lcd.h>
#endif
//functions
void compute_crc_ping(unsigned char *data);
int crc8(unsigned char a);
int compute_crc_data(unsigned char* data, unsigned char dlength);
void packet_handler(const unsigned char* data, unsigned char length);
unsigned char make_ping(unsigned char rcx_id, unsigned char ack_p, unsigned char ack_r);
int make_data(unsigned char* d, unsigned char dlength, unsigned char rcx_id,
unsigned char comm, unsigned char p1, unsigned char p2, unsigned char p3, unsigned char p4);
int check_crc_data(unsigned char* data, unsigned char dlength);
int comm_control(int argc, char **argv);
int ping_sender (int argc, char* argv[]);
int drive_control(int argc, char *argv[]);
void init_comm();
#ifdef DEBUG_TROPOCOL
void display_packet();
#endif
#endif//__tropocol_h__
```

tropocol.c

```
/*
  tropocol.c
  main program which launches a task for drive comtrol and another one for communication
*/
#include "tropocol.h"
//global variables
volatile long int start_time = 0;
volatile unsigned char data_comm[NBR_BYTES_COMM];
volatile unsigned char data_response[NBR_BYTES_COMM];
volatile unsigned char new_response = 0;
volatile unsigned char data_ping = 0;
volatile unsigned int new_data = 0;
volatile unsigned int data_length = 0;
```

```
volatile unsigned int new_comm = 0;
volatile unsigned char rcx_id = 0;
volatile unsigned char ack_p = 0;
volatile unsigned char ack_r = 0;
/*
   commands :
   0 -> 127     command from pc
   128 -> 255   response from rcx
*/
volatile unsigned char rcx_comm = 0;
volatile unsigned char rcx_p1 = 0;
volatile unsigned char rcx_p2 = 0;
volatile unsigned char rcx_p3 = 0;
volatile unsigned char rcx_p4 = 0;
//functions
const note_t my_note[] = {
  { PITCH_D4,  2 }
};
//function to compute packet crc
void compute_crc_ping(unsigned char *data) {
  //lengh of data packets = 1 byte = 8 bits
  unsigned char i = 7;
  unsigned char k = 0;
  unsigned char d = *data;
  d/=2;
  while(i--) {
    k += d % 2;
    d /= 2;
  }
  *data += k % 2;
}
int crc8(unsigned char a) {
  int res=0;
  int i;
  for(i=0;i<8;i++)
    {
      res+=(a%2);
      a/=2;
    }
  return res%2;
}
int compute_crc_data(unsigned char* data, unsigned char dlength) {
  //lengh of data packets = 6 bytes
  if (dlength != NBR_BYTES_COMM)
    return -1;
  data[0] += ((((crc8(data[1]))*2+crc8(data[2]))*2+crc8(data[3]))*2+crc8(data[4]))*2+crc8(data[5]);
  return 0;
}
//function called when a new packet arrives
void packet_handler(const unsigned char* data, unsigned char length) {
  if(new_data == 0)//data treated -> ready for new data
    {
      memcpy(data_comm, data, length);
      data_length = length;
      new_data = 1;//new data not treated
    }
}
unsigned char make_ping(unsigned char rcx_id, unsigned char ack_p, unsigned char ack_r) {
```

```
  unsigned char d = rcx_id * 32 + ack_p * 8 + ack_r * 2;
  compute_crc_ping (&d);
  return d;
}
//format the data packet
int make_data(unsigned char* d, unsigned char dlength, unsigned char rcx_id,
unsigned char comm, unsigned char p1, unsigned char p2, unsigned char p3, unsigned char p4) {
  if (dlength != NBR_BYTES_COMM)
    return -1;
  d[0] = rcx_id * 32;
  d[1] = comm;
  d[2] = p1;
  d[3] = p2;
  d[4] = p3;
  d[5] = p4;
  compute_crc_data(d, NBR_BYTES_COMM);
  return 0;
}
int check_crc_data(unsigned char* data, unsigned char dlength) {
  //lengh of data packets = 6 bytes
  int j;
  int ok = 1;
  unsigned char temp;
  if (dlength != NBR_BYTES_COMM) {
    return -1;
  }

  temp = data[0];
  for(j=0;j<NBR_BYTES_COMM-1 && ok;j++)
    {
      ok=(crc8(data[NBR_BYTES_COMM-1-j])==(temp%2));
      temp/=2;
    }
  return ok? 0 : -1;
}
int comm_control(int argc, char **argv) {
  wakeup_t wait_for_data(wakeup_t data) {
    return new_data;
  }
  while(1) {
    wait_event(wait_for_data, 0);
    //do decoder job
    /*packet format - ping
    rcx_id ack_p ack_r crc
    x x x  x x   x x   x
    packet format - data
    rcx_id crc        data
    x x x  x x x x x  x x x x ....x x x x
    */
    //decode packet
    if (new_comm == 0) {//no processing command
      if (data_length == NBR_BYTES_COMM) {//good length of data packet

        if (check_crc_data(data_comm, data_length) == 0) {//consistent packet

          rcx_id = data_comm[0] / 32;
          if (rcx_id == RCX_ID) {//good rcx id
```

```
            if (data_comm[1] < 128) {//command number valid

              rcx_comm = data_comm[1] ;
              rcx_p1 = data_comm[2] ;
              rcx_p2 = data_comm[3] ;
              rcx_p3 = data_comm[4] ;
              rcx_p4 = data_comm[5] ;
              new_comm = 1 ;

              //send an ack
              ack_p = (ack_p+1) % 4 ;//update ack from pc
              data_ping = make_ping(rcx_id,ack_p,ack_r) ;
              while (lnp_integrity_write(&data_ping, 1) != 0)
                msleep(COLLISION_TIME) ;

              //update start_time for the ping timer
              start_time = sys_time + (RCX_ID *  COLLISION_TIME_COEF2) ;
            }
            else {//command number of response and not a data packet
              //discard packet
            }
          }
          else {//bad rcx id
            //discard packet
          }
        }
        else {//corrupted packet
          //discard packet
        }
      }
      else {//bad length packet
        //discard packet
      }
    }
    else {//processing command
      //discard packet
    }
    new_data = 0 ;
  }
}
int ping_sender (int argc, char* argv[]) {

  wakeup_t wait_for_timer(wakeup_t data) {

    return (((sys_time - start_time) >= PING_INTERVAL) ? 1 : 0) ;
  }
  while(1) {
    wait_event(wait_for_timer, 0) ;

    if (data_ping == 0) {//first ping to send
      data_ping = make_ping(RCX_ID, 0, 0) ;
    }
#ifdef DEBUG_TROPOCOL
    lcd_clear() ;
    lcd_int((int) data_ping) ;
#endif
    while (lnp_integrity_write(&data_ping, 1) != 0) {//send the last ping
      msleep(COLLISION_TIME) ;
```

```
    }

    //reset the timer
    start_time = sys_time;
  }
}
#ifdef DEBUG_TROPOCOL
void display_packet() {
  lcd_clear();
  lcd_int((int) rcx_comm);
  sleep(2);
  lcd_clear();
  lcd_int((int) rcx_p1);
  sleep(2);
  lcd_clear();
  lcd_int((int) rcx_p2);
  sleep(2);
  lcd_clear();
  lcd_int((int) rcx_p3);
  sleep(2);
  lcd_clear();
  lcd_int((int) rcx_p4);
  sleep(2);
  lcd_clear();
}
#endif//DEBUG_TROPOCOL
int drive_control(int argc, char *argv[]){
  wakeup_t wait_for_comm(wakeup_t data) {
    return new_comm;
  }
  int temp = 0, temp2 = 0, temp3 = 0;
  while(1) {
    //do fun things
    wait_event(wait_for_comm, 0);
    //new command to treat
    //clear screen if stall
    cls();
    cputs("id 3");
    switch (rcx_comm) {
    case 1 : //skeleton of ask command

      //make and send the response

      //format and send the ping following the response
      ack_r = (ack_r + 1) % 4;
      data_ping = make_ping(rcx_id, ack_p, ack_r);
      while (lnp_integrity_write(&data_ping, 1) != 0) {//send the last ping
        msleep(COLLISION_TIME);
      }

      //update start_time for the ping timer
      start_time = sys_time + (RCX_ID *  COLLISION_TIME_COEF2);
      new_response = 1;
      break;
    case 2 : //skeleton of sendorder command

      //apply command
```

```
      new_response = 0 ;
      break ;
    default ://unknown command between 128 -> 255
      //discard command because unknown command
#ifdef DEBUG_TROPOCOL
      display_packet() ;
#endif DEBUG_TROPOCOL
      break ;
    }
    new_comm = 0 ;
  }
}
void init_comm() {
  //display the RCX_ID
  cls() ;
  cputs("id 3") ;
  lnp_integrity_set_handler(packet_handler) ;
  lnp_logical_range(1) ;
  start_time = sys_time + (RCX_ID *  COLLISION_TIME_COEF2) ;
}
```

D.2 Ordinateur client

Cette partie du code pour le protocole de communication fut développée par S. Collette, à l'occasion du Printemps des Sciences [32].

D.2.1 Pseudocode

Algorithme 8 Pseudocode du logiciel de protocole de communication du côté de l'ordinateur client. Chaque ordinateur client gère une tour d'émission/réception infra-rouge.

ORDINATEURCLIENT
  Initialiser les structures de données
  Connexion au serveur TROPOCOLPC
  Lancer la tâche Emission PC→RCX
  Lancer la tâche EMISSION RCX←PC
ENDORDINATEURCLIENT

THREAD Emission PC→RCX
  WHILE tâche active
    Attendre une donnée à envoyer
    Transmettre la donnée à *LNPD* (vers le RCX)
  ENDWHILE
ENDTHREAD

THREAD EMISSION RCX←PC
  WHILE tâche active
    Attendre une donnée de *LNPD* (du RCX)
    Vérifier la validité du CRC et la longueur de la trame
    Transmettre la donnée au serveur TROPOCOL
  ENDWHILE
ENDTHREAD

LNPD [28] est un démon tournant sur les ordinateurs clients et permettant de transmettre des bits par la tour d'émission/réception infra-rouge.

D.2.2 Code Source

towermaster.c

```
/*towermaster.c fait par Sebastien Collette secollet@ulb.ac.be
  Vous êtes ici dans le TowerMaster (TM).
  C'est une sorte de démon qui, avec son ami LNPd, s'occupe de gérer une tour...
  Usage : towermaster PORT_SERVEUR IP_LNP PORT_LNP
*/
#include <stdlib.h>
#include "socket.h"
#include <liblnp.h>
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <pthread.h>
#include <signal.h>
#include <errno.h>
int socketdatamaster ;
int socketpingmaster ;
int socketdata ;
int socketping ;
int connected=0 ;
pthread_t handlerthread ;
int CRC8(unsigned char a)
{
  int res=0 ;
  int i ;
  for(i=0 ;i<8 ;i++)
    {
      res+=(a%2) ;
      a/=2 ;
    }
  return res%2 ;
}
void int_handler(const unsigned char* data,unsigned char length)
{
  if(connected)
    {
      printf("Transmission RCX->PC\n") ;
      if(length==1)
        {
          int temp=*data ;
          printf("Ping reçu %i\n",temp) ;
          if( !CRC8(*data))
            {
              int rcxorig ;
              int ackpc ;
              int ackrcx ;
              rcxorig=temp/32 ;
              printf("Ping du RCX n°%i\n",rcxorig) ;
              ackpc=(temp/2)%4 ;
              ackrcx=(temp/8)%4 ;
              printf("ackrcx : %i ackpc : %i\n",ackpc,ackrcx) ;
              SocketWrite(socketping,data,length) ;
              printf("Ping transmis\n") ;
            }
          else
```

```
            {
              printf("Ping : mauvais CRC\n") ;
            }
        }
      else if((length==6) && (data[1]>127))
        {
          int temp=data[0] ;
          int j ;
          int ok=1 ;
          int rcxorig=temp/32 ;

          printf("Data reçu du RCX n°%i !\n",rcxorig) ;
          printf("Données du paquet : ") ;
          for(j=0 ;j<5 && ok ;j++)
            {
              printf("%i ",(int) data[j+1]) ;
              ok=(CRC8(data[5-j])==(temp%2)) ;
              temp/=2 ;
            }
          printf("\n") ;
          if( !ok)
            {
              int l ;
              int m ;
              printf("CRC incorrect : ") ;
              for(l=0 ;l<6 ;l++)
                {
                  m=data[l] ;
                  printf("%i ",m) ;
                }
              printf("\n") ;
            }
          else
            {
              SocketWrite(socketdata,data,length) ;
              printf("Data transmis\n") ;
            }
        }
      else if(length !=6)
        {
          printf("Trame erronée reçue : longueur inconnue...\n") ;
        }
    }
}
void handler_manager()
{
  lnp_integrity_set_handler(int_handler) ;
  while(1)
    sleep(1) ;
}
int main(int argc, char** argv)
{
  int port_serveur,port_lnp ;
  unsigned char buf[6] ;
  sscanf(argv[1],"%i",&port_serveur) ;
  sscanf(argv[3],"%i",&port_lnp) ;
  socketdatamaster=SocketServer(port_serveur) ;
  socketpingmaster=SocketServer(port_serveur+1) ;
```

```
    socketdata=ServerWait(socketdatamaster) ;
    socketping=ServerWait(socketpingmaster) ;
    if(lnp_init(argv[2],port_lnp,0,0,0))
      {
        perror("Problème LNP...\n") ;
        exit(1) ;
      }
    pthread_create(&handlerthread,NULL,handler_manager,NULL) ;

    while(1)
      {
        int retval ;
        connected=1 ;
        while(SocketReadSafe(socketdata,buf,6)>=0)
          {
            int t ;
            printf("Transmission PC->RCX\n") ;
            printf("Données : ") ;
            for(t=1 ;t<6 ;t++)
              printf("%i ",(int)buf[t]) ;
            printf("\n") ;
            while(lnp_integrity_write(buf,6))
              ;
          }
        printf("Déconnexion détectée...\n") ;
        connected=0 ;

        SocketStop(socketdata) ;
        SocketStop(socketping) ;
        socketdata=ServerWaitSafe(socketdatamaster) ;
        socketping=ServerWaitSafe(socketpingmaster) ;
      }
    SocketStopsocketdatamaster) ;
    SocketStop(socketpingmaster) ;
  }
```

D.3 Ordinateur Central

Cette partie du code pour le protocole de communication fut développée par S. Collette, à l'occasion du Printemps des Sciences [32].

D.3.1 Pseudocode

Algorithme 9 Pseudocode du logiciel de protocole de communication du côté de l'ordinateur central. Il gère tous les ordinateurs clients dédiés au protocole de communication.

```
TROPOCOLPC
  Lire fichier de configuration
  FOR tous les ORDINATEURCLIENT
    Créer une connexion
    Initialiser les structures de données associées
  ENDFOR
  Lancer la tâche PINGMANAGER
ENDTROPOCOLPC

THREAD PINGMANAGER
  WHILE tâche active
    Attendre un ping des ORDINATEURCLIENT
    IF ping valide THEN
      Mettre à jour la localisation du RCX considéré
      Mettre à jour les numéros de séquence
      IF numéro de séquence inattendu du à une perte de données THEN
        Renvoyer le paquet de données perdu (buffer EMISSION)
      ENDIF
      IF numéro de séquence inattendu du à un envoi RCX→PC THEN
        IF Si donnée disponible THEN
          Mettre la donnée reçue du ORDINATEURCLIENT dans le buffer RECEPTION
          Mettre à jour le numéro de séquence
        ELSE
          Demander le réenvoi au RCX (via SENDORDER spécifique)
        ENDIF
      ENDIF
    ENDIF
  ENDWHILE
ENDTHREAD

FUNCTION SENDORDER
  Trouver la tour correspondant au RCX considéré
  Envoyer la donnée vers le ORDINATEURCLIENT correspondant
  Mettre à jour le numéro de séquence attendu
  Sauver la donnée dans le buffer EMISSION
ENDFUNCTION

FUNCTION ASKVALUE
  Trouver la tour correspondant au RCX considéré
  Envoyer la donnée vers le ORDINATEURCLIENT correspondant
  Mettre à jour le numéro de séquence attendu
  Sauver la donnée dans le buffer EMISSION
  Attendre la disponibilité de données dans le buffer RECEPTION
  Retourner les données reçues
ENDFUNCTION
```

D.3.2 Code Source

tropocolpc.h

```
/* tropocolpc.h fait par Sebastien Collette secollet@ulb.ac.be
  Interface transparente au système multi-tours. Gestion de plusieurs RCX via plusieurs tours.
  ATTENTION : la gestion telle qu'implémentée ne permet pas d'exécuter plusieurs instances de ce code
  au même moment. Je veux dire par là que une et UNE SEULE tache doit interfacer avec le programme,
  ou, dans le cas contraire, aucun appel concurrent ne peut être effectué.
*/
#ifndef __tropocol_h__
#define __tropocol_h__
// CONFIGURATION -----------------------------------------------------------------
#define MAX_TOWER 8 //Attention aux effets de bord.
#define MAX_RCX 8 //Attention aux effets de bord (format de trame)!!!
#define SENDWAIT 4 //Timings, ne pas modifier sans tests preallables!
#define RECEIVEWAIT 41
#define LOOPWAIT 20000
/*
  Les define possibles sont ERRORMSG, DEBUG et DEBUGMORE. Affichage des messages d'erreur, de debug ou de
  Pour un debug detaillé, mettre DEBUG et DEBUGMORE.
*/
#define ERRORMSG

//-------------------------------------------------------------------------------
void init_tropocol(char* configfile);
void send_order(unsigned char rcx, unsigned char cmd, unsigned char a,
unsigned char b, unsigned char c, unsigned char d);
int ask_value(unsigned char rcx, unsigned char *cmd, unsigned char *a,
unsigned char *b, unsigned char *c, unsigned char *d);
void end_tropocol();
void cellular_debug();
void add_rcx(int num); //Permet également de faire un reset du rcx choisi.
void del_rcx(int num);
#endif //__tropocol_h__
```

tropocolpc.c

```
/* tropocolpc.c fait par Sebastien Collette secollet@ulb.ac.be */
#include "tropocol.h"
#include "socket.h"
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include <signal.h>
int numtowers; //Nombre de tours effectivement utilisées
int toweractu[MAX_RCX]; //Cellule courante
int ackpc[MAX_RCX];
int ackrcx[MAX_RCX];
int rcx[MAX_RCX];
int socketdatalist[MAX_TOWER];
int socketpinglist[MAX_TOWER];
unsigned char pendingdata[MAX_RCX][6];
unsigned char pendingbackdata[MAX_RCX][6];
volatile int pendingflag[MAX_RCX];
int pendingackpc[MAX_RCX];
```

```
int pendingackrcx[MAX_RCX] ;
volatile int pendingbackflag[MAX_RCX] ;
volatile int pendingbackackflag[MAX_RCX] ;
pthread_t pingthread ;
//-----  Variables permettant le débug et les statistiques  -----
int hopcount=0 ;
//-------------------------------------------------------------
void ping_manager()
{
  fd_set rfds ;
  struct timeval tv ;
  int retval ;
  int i ;
  while(1)
    {
      int n=0 ;
#ifdef DEBUGMORE
      printf("numtowers : %i\n",numtowers) ;
#endif
      FD_ZERO(&rfds) ;
      for(i=0 ;i<numtowers ;i++)
        {
          FD_SET(socketpinglist[i],&rfds) ;
          if(socketpinglist[i]>n)
            n=socketpinglist[i] ;
          FD_SET(socketdatalist[i],&rfds) ;
          if(socketdatalist[i]>n)
            n=socketdatalist[i] ;
        }
      n++ ;

      tv.tv_sec=1 ;
      tv.tv_usec=0 ;
#ifdef DEBUGMORE
      printf("Le select commence...\n") ;
#endif
      retval=select(n,&rfds,NULL,NULL,&tv) ;
#ifdef DEBUGMORE
      printf("select retval : %i\n",retval) ;
      printf("Le select se termine...\n") ;
#endif
      if(retval)
        {
          for(i=0 ;i<numtowers && !FD_ISSET(socketdatalist[i],&rfds) ;i++)
            ;
          if(i<numtowers)
            {
              unsigned char buf[6] ;
              int rcxorig ;
#ifdef DEBUG
              printf("Paquet reçu...\n") ;
#endif
              if(SocketRead(socketdatalist[i],buf,6)>=0)
                {
                  rcxorig=buf[0]/32 ;
#ifdef DEBUG
                  printf("Provenance : %i\n",rcxorig) ;
#endif
```

```
                    if(pendingbackackflag[rcxorig])
                      {
                        memcpy(pendingbackdata[rcxorig],buf,6) ;
                        pendingbackflag[rcxorig]=1 ;
                        pendingbackackflag[rcxorig]=0 ;
                      }
                    else
                      {
#ifdef DEBUG
                        printf("Duplicat détecté...\n") ;
#endif
                      }
                  }
              }
            for(i=0 ;i<numtowers && !FD_ISSET(socketpinglist[i],&rfds) ;i++)
              ;
            if(i<numtowers)
              {
                unsigned char lu ;
                int rcxorig ;
#ifdef DEBUGMORE
                printf("Certaines tours ont reporté un ping...\n") ;
#endif
                if(SocketRead(socketpinglist[i],&lu,1)>=0)
                  {
                    rcxorig=lu/32 ;
#ifdef DEBUGMORE
                    printf("Ping du RCX n°%i\n",rcxorig) ;
#endif
                    if(i !=toweractu[rcxorig])
                      hopcount++ ;
                    toweractu[rcxorig]=i ;
                    ackrcx[rcxorig]=(lu/2)%4 ;
                    ackpc[rcxorig]=(lu/8)%4 ;
#ifdef DEBUGMORE
                    printf("ackrcx : %i ackpc : %i\n") ;
#endif

                    if(pendingflag[rcxorig])
                      {
                        if(ackpc[rcxorig] !=pendingackpc[rcxorig])
                          {
                            if(pendingflag[rcxorig]>1)
                              pendingflag[rcxorig]-- ;
                            else
                              {
#ifdef ERRORMSG
                                printf("Erreur de numéro de séquence, réenvoi...\n") ;
#endif
                                resend_order(rcxorig,pendingdata[rcxorig]) ;
                                pendingflag[rcxorig]=SENDWAIT ;
                              }
                          }
                        else
                          {
#ifdef DEBUG
                            printf("Numéro de séquence confirmé...\n") ;
#endif
```

```
                        pendingflag[rcxorig]=0 ;
                      }
                  }
                else
                  if(pendingbackackflag[rcxorig]) //Si on attends un ack de
RCX->PC && donnée pas reçue
                    if(ackrcx[rcxorig]==pendingackrcx[rcxorig]) //Si numéro
de séquence déjà atteint
                      {
#ifdef ERRORMSG
                        printf("Demande de réenvoi, donnée non-reçue !\n") ;
#endif
                        send_order(rcxorig,127,0,0,0,0) ;
                      }
                    else //Si numéro de séquence pas encore atteint, pas de
stress.
                      pendingbackackflag[rcxorig]-- ; //On décrémente pour
éviter un lock
              }
          }
      }
    else
      {
        printf("Aucun RCX accessible...\n") ;
      }
  }
}
void init_tropocol(char* configfile)
{
  int i ;
  FILE *config ;

  config = fopen(configfile,"r") ;
  fscanf(config,"%i",&numtowers) ;
  for(i=0 ;i<MAX_RCX ;i++)
    {
      rcx[i]=0 ;
      pendingflag[i]=0 ;
      pendingbackflag[i]=0 ;
      pendingbackackflag[i]=0 ;
    }
#ifdef DEBUG
  printf("Tentative de connexion aux %i tours...\n",numtowers) ;
#endif
  for(i=0 ;i<numtowers ;i++)
    {
      char ip[255] ;
      int port ;
      fscanf(config,"%s%i",ip,&port) ;
#ifdef DEBUG
      printf("ip : %s :%i\n",ip,port) ;
#endif
      if((socketdatalist[i]=SocketClient(ip,port))<0)
        {
          printf("Erreur de connexion\n") ;
          exit(1) ;
        }
      sleep(1) ;
```

```
      if((socketpinglist[i]=SocketClient(ip,port+1))<0)
        {
          printf("Erreur de connexion\n") ;
          exit(1) ;
        }
      //printf("Socketping[%i]=%i\n",i,socketpinglist[i]) ;
#ifdef DEBUG
      printf("Tour %i connectée.\n",i) ;
#endif
    }
  fclose(config) ;
  sleep(1) ;
  pthread_create(&pingthread,NULL,ping_manager,NULL) ;
}
int CRC8(unsigned char a)
{
  int res=0 ;
  int i ;
  for(i=0 ;i<8 ;i++)
    {
      res+=(a%2) ;
      a/=2 ;
    }

  return res%2 ;
}
void resend_order(unsigned char rcx, char *buf)
{
  SocketWrite(socketdatalist[toweractu[rcx]],buf,6) ;
}
void send_order(unsigned char rcx, unsigned char cmd, unsigned char a,
unsigned char b, unsigned char c, unsigned char d)
{
  unsigned char buf[6] ;
  int i ;
  printf("send order %i %i %i %i %i %i\n",(int)rcx, (int)cmd, (int)a, (int)b, (int)c, (int)d) ;
  while(pendingflag[rcx])
    usleep(LOOPWAIT) ;
  buf[0]=((((rcx*2+CRC8(cmd))*2+CRC8(a))*2+CRC8(b))*2+CRC8(c))*2+CRC8(d) ;
#ifdef DEBUG
  printf("Le CRC du paquet à envoyer est : %i\n",(int)buf[0]) ;
#endif

  buf[1]=cmd ;
  buf[2]=a ;
  buf[3]=b ;
  buf[4]=c ;
  buf[5]=d ;
  pendingackpc[rcx]=(ackpc[rcx]+1)%4 ;
  memcpy(pendingdata[rcx],buf,6) ;
  SocketWrite(socketdatalist[toweractu[rcx]],buf,6) ;
  pendingflag[rcx]=SENDWAIT ;
}
int ask_value(unsigned char rcx, unsigned char *cmd, unsigned char *a,
unsigned char *b, unsigned char *c, unsigned char *d)
{
  unsigned char buf[6] ;
  int i ;
```

```
  int j ;
  while(pendingflag[rcx])
    usleep(LOOPWAIT) ;
  buf[0]=((((rcx*2+CRC8(*cmd))*2+CRC8(*a))*2+CRC8(*b))*2+CRC8(*c))*2+CRC8(*d) ;
  buf[1]=(*cmd) ;
  buf[2]=(*a) ;
  buf[3]=(*b) ;
  buf[4]=(*c) ;
  buf[5]=(*d) ;
  pendingackpc[rcx]=(ackpc[rcx]+1)%4 ;
  pendingackrcx[rcx]=(ackrcx[rcx]+1)%4 ;
  memcpy(pendingdata[rcx],buf,6) ;
  SocketWrite(socketdatalist[toweractu[rcx]],buf,6) ;
  pendingflag[rcx]=SENDWAIT ;
  pendingbackackflag[rcx]=RECEIVEWAIT ; //Le délai n'a pas d'importance : seul
un plantage du RCX engendre une perte.
#ifdef DEBUG
  printf("Attente d'un paquet RCX->PC.\n") ;
#endif
  while( !pendingbackflag[rcx] && pendingbackackflag[rcx])
    usleep(LOOPWAIT) ;

  if( !pendingbackackflag[rcx] && !pendingbackflag[rcx])
    {
      printf("RCX inaccessible ou planté pendant une commande
ask_value...\n") ;
      return -1 ;
    }
  else
    {
      *cmd=pendingbackdata[rcx][1] ;
      *a=pendingbackdata[rcx][2] ;
      *b=pendingbackdata[rcx][3] ;
      *c=pendingbackdata[rcx][4] ;
      *d=pendingbackdata[rcx][5] ;
      pendingbackflag[rcx]=0 ;
      while(pendingackrcx[rcx] !=ackrcx[rcx])
        usleep(LOOPWAIT) ;
    }
  return 0 ;
}
void end_tropocol()
{
  int i ;
  for(i=0 ;i<numtowers ;i++)
    {
      SocketStop(socketdatalist[i]) ;
      SocketStop(socketpinglist[i]) ;
    }
  pthread_kill(pingthread,SIGKILL) ;
}
void cellular_debug()
{
  int i ;
  for(i=0 ;i<MAX_RCX ;i++)
    if(rcx[i])
      {
        printf("RCX %i\n",i) ;
```

```
            printf("Tour actuelle : %i\n",toweractu[i]) ;
            printf("ack RCX : %i\n",ackrcx[i]) ;
            printf("ack PC : %i\n",ackpc[i]) ;
            printf("\n") ;
          }
      printf("Hopcount : %i\n",hopcount) ;
    }
    void add_rcx(int num)
    {
      rcx[num]=1 ;
      ackpc[num]=0 ;
      ackrcx[num]=0 ;
    }
    void del_rcx(int num)
    {
      rcx[num]=0 ;
    }
```

Fichier de Configuration : tower.cfg

```
2 //Nombre d'ordinateurs clients
192.168.1.19 //Adresse IP de l'ordinateur client 1
21000 //port de l'ordinateur client 1
127.0.0.1 //Adresse IP de l'ordinateur client 2
21000 //port de l'ordinateur client 2
```

Annexe E

Analyse de Rotations Successives

D'autres exemples d'estimation de la distribution de probabilité et histogramme des mesures de luminosité pour des numéros de rotations différents. Les figures E.1 et E.2 donnent l'estimation et l'histogramme pour les numéros de rotation 8 et 54 respectivement.

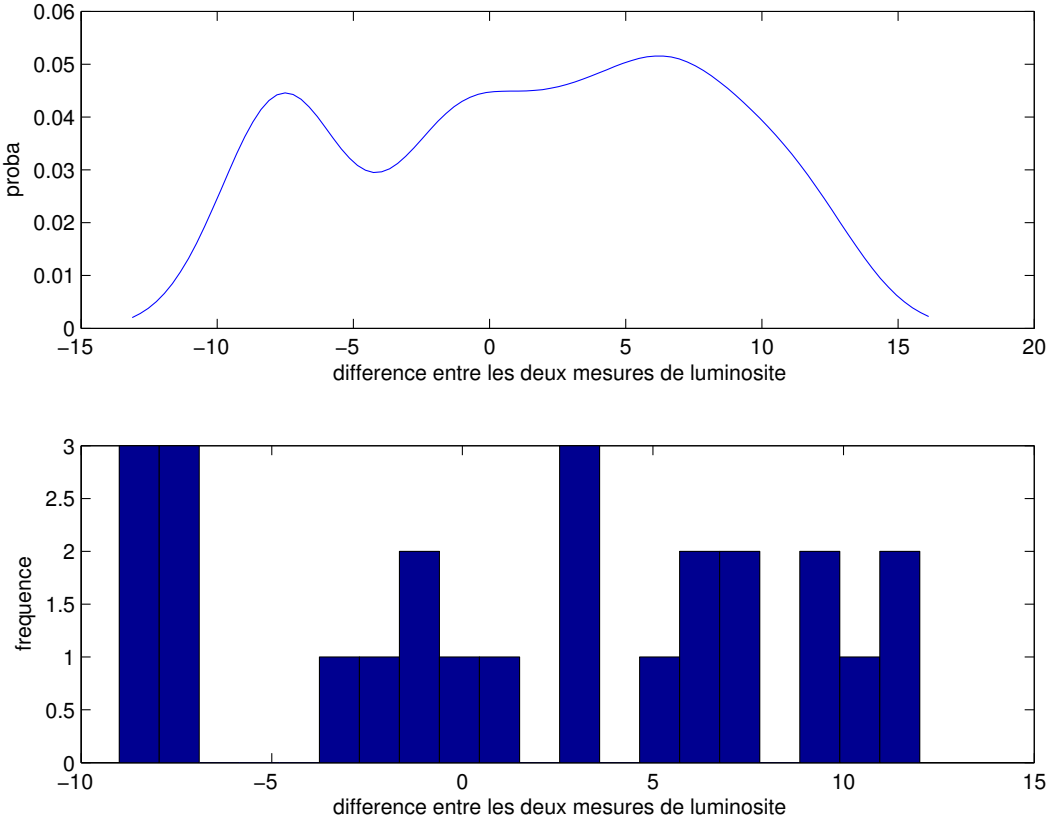


FIG. E.1 – Estimation de la distribution de probabilité et histogramme des mesures de luminosité pour la rotation numéro 8.

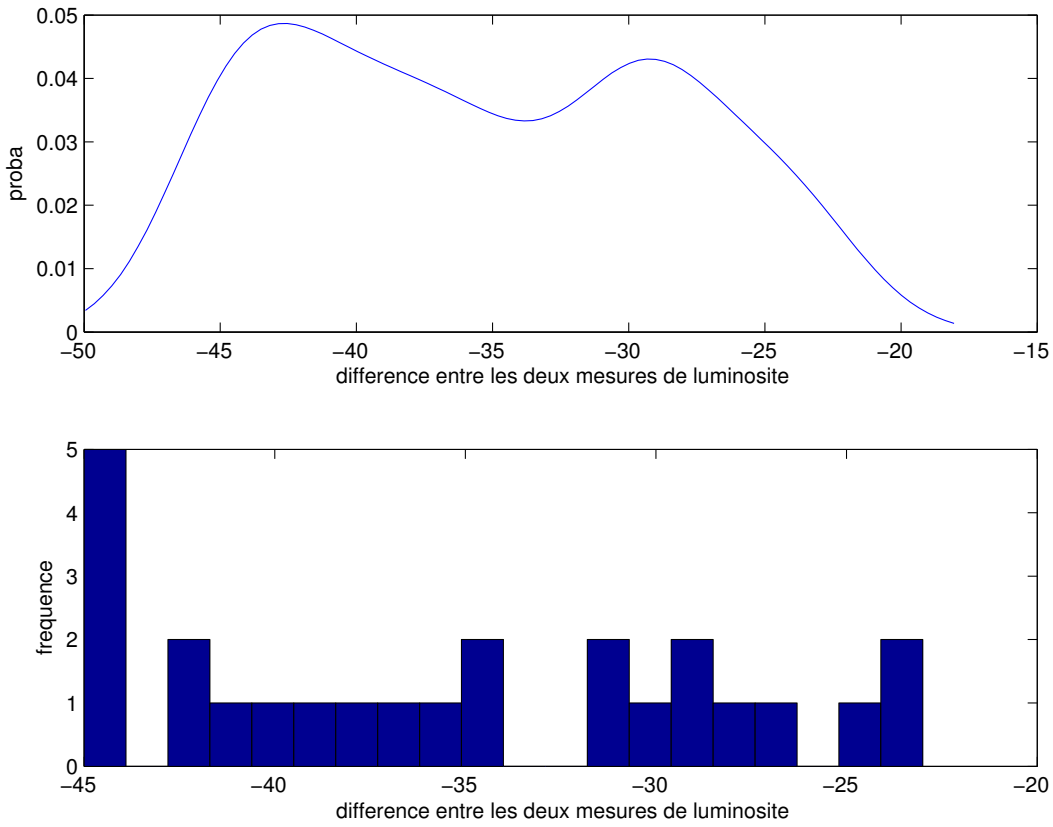


FIG. E.2 – Estimation de la distribution de probabilité et histogramme des mesures de luminosité pour la rotation numéro 54.

Annexe F

Test de Permutation

F.1 Procédure

Dans le cadre des tests de permutation concernant les distributions des deux senseurs, la procédure est décrite en détail ci-dessous.

Soit $a_1, a_2, ..., a_N$ un échantillon aléatoire de la distribution inconnue $a \sim x_1$ et $b_1, b_2, ..., b_M$ un échantillon aléatoire de la distribution inconnue $b \sim x_2$. M et N sont fixés à 25 en accord avec les données collectées par l'expérience précédente.

Soit D_{M+N} l'ensemble des $M + N$ données observées.

Soit R l'ensemble des permutations calculées. $R = \begin{pmatrix} M+N \\ M \end{pmatrix}$ si toutes les permutations sont considérées. Cependant, ce nombre est trop important pour M et N valant 25. Nous ne considérerons qu'un nombre arbitraire[1] de permutations.

Soit s une statistique telle que $s(D_{M+N}) = \hat{\mu}_1 - \hat{\mu}_2$ avec $\hat{\mu}_1$ et $\hat{\mu}_2$ respectivement la moyenne empirique sur $D_M$et D_N. L'hypothèse nulle est donc $H : \hat{\mu}_1 = \hat{\mu}_2$ et l'hypothèse alternative $\bar{H} : \hat{\mu}_1 \neq \hat{\mu}_2$.

Procédure des Tests de Permutation

1. Calcul de la statistique s avec l'ensemble original des données afin d'obtenir $s^* = s(D)$.
2. Pour chaque $i^{ème}$ permutation de l'ensemble de données original noté D_i avec $i = 1, 2, ..., R$, calculer $s_i = s(D_i)$.
3. Si la valeur s^* tombe dans la queue $\alpha/2$ de la distribution des s_i, l'hypothèse nulle est rejetée avec une erreur α de type[2] I.

F.2 Résultats

Voici le résultat des tests de permutation sur un jeu de données constitué de 55 rotations d'amplitude unitaire avec 25 prises de mesures de luminosité à chaque rotation :

```
rotation 1 (zone obscure) : hypothese rejetee
rotation 2 (zone obscure) : hypothese rejetee
rotation 3 (zone obscure) : hypothese rejetee
rotation 4 (zone obscure) : hypothese rejetee
rotation 5 (zone obscure) : hypothese rejetee
rotation 6 (zone obscure) : hypothese rejetee
rotation 7 (zone obscure) : hypothese rejetee
```

[1] R est fixé empiriquement à 1500. En effet, cela représente un bon compromis entre temps de calcul et précision.

[2] L'erreur de type I est l'erreur qui est faite lorsque l'hypothèse nulle est rejetée si elle est correcte. Le niveau de signification α est la probabilité de faire une erreur de type I.

```
rotation 8 (zone obscure) : hypothese non rejetee
rotation 9 (zone obscure) : hypothese rejetee
rotation 10 (zone obscure) : hypothese rejetee
rotation 11 (zone obscure) : hypothese rejetee
rotation 12 (zone obscure) : hypothese rejetee
rotation 13 (zone obscure) : hypothese rejetee
rotation 14 (zone obscure) : hypothese rejetee
rotation 15 (zone obscure) : hypothese non rejetee
rotation 16 (zone obscure) : hypothese rejetee
rotation 17 (zone obscure) : hypothese rejetee
rotation 18 (zone obscure) : hypothese rejetee
rotation 19 (zone obscure) : hypothese non rejetee
rotation 20 (zone obscure) : hypothese rejetee
rotation 21 (zone obscure) : hypothese rejetee
rotation 22 (zone obscure) : hypothese rejetee
rotation 23 (zone lumineuse) : hypothese rejetee
rotation 24 (zone lumineuse) : hypothese rejetee
rotation 25 (zone lumineuse) : hypothese rejetee
rotation 26 (zone lumineuse) : hypothese rejetee
rotation 27 (zone lumineuse) : hypothese rejetee
rotation 28 (zone lumineuse) : hypothese rejetee
rotation 29 (zone lumineuse) : hypothese rejetee
rotation 30 (zone lumineuse) : hypothese rejetee
rotation 31 (zone lumineuse) : hypothese rejetee
rotation 32 (zone lumineuse) : hypothese rejetee
rotation 33 (zone lumineuse) : hypothese rejetee
rotation 34 (zone lumineuse) : hypothese rejetee
rotation 35 (zone obscure) : hypothese rejetee
rotation 36 (zone obscure) : hypothese rejetee
rotation 37 (zone obscure) : hypothese rejetee
rotation 38 (zone obscure) : hypothese rejetee
rotation 39 (zone obscure) : hypothese rejetee
rotation 40 (zone obscure) : hypothese rejetee
rotation 41 (zone obscure) : hypothese rejetee
rotation 42 (zone obscure) : hypothese rejetee
rotation 43 (zone obscure) : hypothese rejetee
rotation 44 (zone obscure) : hypothese rejetee
rotation 45 (zone obscure) : hypothese rejetee
rotation 46 (zone obscure) : hypothese rejetee
rotation 47 (zone obscure) : hypothese rejetee
rotation 48 (zone obscure) : hypothese rejetee
rotation 49 (zone obscure) : hypothese rejetee
rotation 50 (zone obscure) : hypothese rejetee
rotation 51 (zone obscure) : hypothese rejetee
rotation 52 (zone obscure) : hypothese rejetee
rotation 53 (zone obscure) : hypothese rejetee
rotation 54 (zone obscure) : hypothese rejetee
rotation 55 (zone obscure) : hypothese rejetee
```

Seules des rotations en zone obscure passent le test avec succès. Aucune position en face de la source lumineuse ne peut donc être trouvée par cette technique.

F.3 Statistique Modifiée

Nous avons vu que la statistique précédemment choisie ne permet pas de trouver une position en face de la source lumineuse. Cependant, nous pouvons remarquer empiriquement que de telles

positions sont caractérisées par une différence entre les deux senseurs de luminosité inférieure à LC fixée à 40. La statistique suivante permet d'utiliser les tests de permutation pour trouver ces positions :

$$s(D_i) = \begin{cases} 0 & si\ |\hat{\mu}_1 - \hat{\mu}_2| \leq LC \\ (\hat{\mu}_1 - \hat{\mu}_2) - LC & si\ (\hat{\mu}_1 - \hat{\mu}_2) > LC \\ (\hat{\mu}_1 - \hat{\mu}_2) + LC & si\ (\hat{\mu}_1 - \hat{\mu}_2) < -LC \end{cases} \quad \text{(F.1)}$$

F.3.1 Résultats

```
rotation 1 (zone obscure) : hypothese non rejetee
rotation 2 (zone obscure) : hypothese non rejetee
rotation 3 (zone obscure) : hypothese non rejetee
rotation 4 (zone obscure) : hypothese non rejetee
rotation 5 (zone obscure) : hypothese non rejetee
rotation 6 (zone obscure) : hypothese non rejetee
rotation 7 (zone obscure) : hypothese non rejetee
rotation 8 (zone obscure) : hypothese non rejetee
rotation 9 (zone obscure) : hypothese non rejetee
rotation 10 (zone obscure) : hypothese non rejetee
rotation 11 (zone obscure) : hypothese non rejetee
rotation 12 (zone obscure) : hypothese non rejetee
rotation 13 (zone obscure) : hypothese non rejetee
rotation 14 (zone obscure) : hypothese non rejetee
rotation 15 (zone obscure) : hypothese non rejetee
rotation 16 (zone obscure) : hypothese non rejetee
rotation 17 (zone obscure) : hypothese rejetee
rotation 18 (zone obscure) : hypothese non rejetee
rotation 19 (zone obscure) : hypothese non rejetee
rotation 20 (zone obscure) : hypothese non rejetee
rotation 21 (zone obscure) : hypothese non rejetee
rotation 22 (zone obscure) : hypothese rejetee
rotation 23 (zone lumineuse) : hypothese rejetee
rotation 24 (zone lumineuse) : hypothese rejetee
rotation 25 (zone lumineuse) : hypothese rejetee
rotation 26 (zone lumineuse) : hypothese rejetee
rotation 27 (zone lumineuse) : hypothese rejetee
rotation 28 (zone lumineuse) : hypothese non rejetee
rotation 29 (zone lumineuse) : hypothese rejetee
rotation 30 (zone lumineuse) : hypothese rejetee
rotation 31 (zone lumineuse) : hypothese rejetee
rotation 32 (zone lumineuse) : hypothese rejetee
rotation 33 (zone lumineuse) : hypothese rejetee
rotation 34 (zone lumineuse) : hypothese rejetee
rotation 35 (zone obscure) : hypothese rejetee
rotation 36 (zone obscure) : hypothese non rejetee
rotation 37 (zone obscure) : hypothese non rejetee
rotation 38 (zone obscure) : hypothese non rejetee
rotation 39 (zone obscure) : hypothese non rejetee
rotation 40 (zone obscure) : hypothese non rejetee
rotation 41 (zone obscure) : hypothese rejetee
rotation 42 (zone obscure) : hypothese rejetee
rotation 43 (zone obscure) : hypothese rejetee
rotation 44 (zone obscure) : hypothese rejetee
rotation 45 (zone obscure) : hypothese rejetee
rotation 46 (zone obscure) : hypothese rejetee
rotation 47 (zone obscure) : hypothese rejetee
rotation 48 (zone obscure) : hypothese rejetee
rotation 49 (zone obscure) : hypothese non rejetee
```

```
rotation 50 (zone obscure) : hypothese non rejetee
rotation 51 (zone obscure) : hypothese non rejetee
rotation 52 (zone obscure) : hypothese non rejetee
rotation 53 (zone obscure) : hypothese non rejetee
rotation 54 (zone obscure) : hypothese non rejetee
rotation 55 (zone obscure) : hypothese non rejetee
```

Il existe de nombreuses rotations passant le test de permutation avec la statistique modifiée mais une seule est en zone lumineuse. Cette position est donc en face de la source lumineuse. Cette technique fonctionne dans la majorité des cas mais elle est très sensible au paramètre LC. En effet, plus LC est petit, plus la position trouvée est en face de la lumière. Néanmoins, la probabilité de ne trouver aucune rotation passant le test de permutation est inversement proportionnelle à la valeur de LC. Par exemple, si LC vaut 5, la probabilité de trouver une position en face de la source lumineuse est très faible. Inversement, si LC vaut 100, de nombreuses positions seront trouvées et aucune information qualitative ne pourra déterminer la meilleure d'entre elles.

F.4 Code Source

permutation_test_sensor.m

```
function [A, T] = permutation_test_sensor(legomat, nb1, nb2, alpha, lim_light, displ, st)
%PERMUTATION_TEST_SENSOR Summary of this function goes here
%  Detailed explanation goes here
%number of measures for the left (M) and the right (N) sensors at each rotation
M = 25 ;
N = 25 ;
%nb is the number of measures take into account
%it allows to search the minimal number of measures to have a sufficient
%result
if nb1 < M && nb1 > 0
    M = nb1 ;
end
if nb2 < N && nb2 > 0
    N = nb2 ;
end
%number of permutation computed
R = 1500 ;
%total number of permutation
n = factorial(M+N) / (factorial(M)*factorial(N)) ;
%take the min between R and n
if R > n
    R = n ;
end
%significance level
alpha = alpha / 100 ;
%constant which allows to distinguish the light and the dark sides
LIMIT_LIGHT_DARK = 650 ;
%used in the tuned statistic to consider an interval of light
LIMIT_LIGHT_DIFF = lim_light ;
%this function compute a certain statistic (mean) for left (V1) and right (V2)
%sensor at each rotation. It allows to determine the type of robot side
[V1, V2] = make_legomat_find_light(legomat, 'mean') ;
%V1(i) contains the mean of the left sensor at rotation i
%V2(i) contains the mean of the right sensor at rotation i
%this function transform the workspace legomat in a matrix to compute the
%permutation test efficiently
T = transform_sensor_distrib(legomat) ;
```

```
%T(i, :) contains all measures from left sensor at rotation i
%T(i+1, :) contains all measures from right sensor at rotation i
for i=1 :2 :size(T,1)-1   %for each rotation

    D = [T(i,1 :M) T(i+1,1 :N)] ;  %sample to shuffle
    %choice of statistic to compute the original one :
    %parameter st = 0 -> simple statistic
    %parameter st = 1 -> tuned statistic
    if st == 0
        t = perm_stat1(D, M, N) ;
    end
    if st == 1
        t = perm_stat2(D, M, N, LIMIT_LIGHT_DIFF) ;
    end
    if st == 2
        t = perm_stat3(D, M, N) ;
    end
    A(round(i/2)) = t ;
    for j=1 :1 :R %do R permutations

        %there is a very small probablility to have twice a same permutation
        %but this is not a great disadvantage
        p = randperm(M+N) ;
        Ds = D(p) ;

        %choice of statistic to compute the statistic of the permutation
        %sample
        if st == 0
            tp(j) = perm_stat1(Ds, M, N) ;
        end
        if st == 1
            tp(j) = perm_stat2(Ds, M, N, LIMIT_LIGHT_DIFF) ;
        end
        if st == 2
            tp(j) = perm_stat3(Ds, M, N) ;
        end
    end

    %compute the percentage of permutation sample statistics < and > the
orginal statistic
    tp=sort(tp) ;
    q_inf=sum(t<=tp)/length(tp) ;
    q_sup=sum(t>=tp)/length(tp) ;
    %display he graph of rotation i
    if round(i/2) == displ

        est = round(R/10) ;
        subplot(2,1,1) ;
        %use ksdensity function with the half of default bandwidth
        [f, x, u] = ksdensity(tp) ;
        [f, x] = ksdensity(tp, 'width', u/2) ;
        plot(x, f) ;
        hold on ;
        %plot the original statistic
        plot(t*ones(1,2), [0 0.5],'r') ;
        title(sprintf('Histogramme de la distribution de la permutation pour
la %ieme rotation', round(i/2))) ;
        xlabel('statistique') ;
```

```
        ylabel('proba') ;
        hold off ;

        subplot(2,1,2) ;
        hist(tp) ;
        hold on ;
        plot(t*ones(1,est+1), 0 :est,'r') ;
        title(sprintf('Histogramme de la distribution de la permutation pour
la %ieme rotation', round(i/2))) ;
        xlabel('statistique') ;
        ylabel('frequence') ;
        hold off ;
    end
    %output for each rotation : info of side and the rejection of
    %hypothesis
    fprintf('rotation %i', round(i/2)) ;
    if V1(round(i/2)) <= LIMIT_LIGHT_DARK || V2(round(i/2)) <=
LIMIT_LIGHT_DARK
        fprintf(' (zone lumineuse) : ') ;
    else
        fprintf(' (zone obscure) : ') ;
    end

    if ((q_inf~=0 || q_sup~=0) && (q_inf<alpha/2 || q_sup<alpha/2))
      fprintf(1,'hypothese rejetee \n')
    else
      fprintf(1,'hypothese non rejetee \n')
    end

end
%simple statistic
function [t] = perm_stat1(D, M, N)
t = mean(D(1 :M)) - mean(D(M+1 :M+N)) ;
%tuned statistic
function [t] = perm_stat2(D, M, N, lim)
t = mean(D(1 :M)) - mean(D(M+1 :M+N)) ;
if abs(t) <= lim
    t = 0 ;
else
    if t < 0
        t = t+lim ;
    else
        t = t-lim ;
    end
end
%tuned statistic
function [t] = perm_stat3(D, M, N)
t = mean(D(1 :M)) - mean(D(M+1 :M+N)) ;
if t < 0
    t = -(1.02^abs(t))+1 ;
else
    t = (1.02^abs(t))-1 ;
end
function [T] = transform_sensor_distrib(legomat)
%load the original data workspace
load(legomat, '-mat') ;
j = -1 ;
k = 1 ;
```

```
for i=1 :1 :length(U1)
    if U1(i) ~= 0 || U2(i) ~=0
        %new series of measures
        j = j+2;
        k = 1;
    end
    T(j,k) = V1(i);
    T(j+1,k) = V2(i);
    k = k+1;
end
function [V1t, V2t] = make_legomat_find_light(legomat, stat)
load(legomat, '-mat');
if length(X1) ~= length(X2) || length(X1) ~= length(U1) || length(X1) ~=
length(U2) || length(X1) ~= length(V1)  || length(X1) ~= length(V2)
    error('the vectors must have the same size!');
end
%number of measures between each step?
i = 2;
while U1(i) == 0
    i = i+1;
end
num_meas = i-1;
for i=0 :1 :((length(U1) / num_meas)-1)
    V1t(i+1) = eval([stat,'(V1(i*num_meas+1 :i*num_meas+num_meas))']);
    V2t(i+1) = eval([stat,'(V2(i*num_meas+1 :i*num_meas+num_meas))']);
end
```

Annexe G

Estimation par Bootstrap

G.1 Procédure

Dans le cadre de l'estimation de la moyenne des différences de luminosité, la technique de *bootstrap* est décrite en détail ci-dessous.

Considérons D_N l'ensemble original des différences contenant N (fixé à 25) différences de luminosité et μ la moyenne réelle de ces différences. La procédure est la suivante :

1. Calcul de la moyenne empirique $\hat{\mu}$ sur D_N.
2. Création d'un ensemble de données $D_{(b)}$ avec $b = 1, 2, ..., R$, obtenu par *bootstrap* en sélectionnant au hasard N données dans D_N et ce, avec remplacement. La probabilité de sélectionner chaque donnée vaut $1/N$. R est le nombre de rééchantillons calculés. Le nombre total de rééchantillons vaut $\begin{pmatrix} 2N-1 \\ N \end{pmatrix}$. Ce nombre est trop important pour permettre un calcul en un temps raisonnable. C'est pourquoi, un nombre arbitraire[1] de rééchantillons seront calculés. La technique de *bootstrap* est donc une méthode de rééchantillonnage stochastique car les rééchantillons sont choisis au hasard.
3. Calcul de la moyenne empirique $\hat{\mu}_{(b)}$ pour chaque ensemble de données $D_{(b)}$ obtenu par *bootstrap*.
4. Le biais de l'estimateur $\hat{\mu}$ est calculé à partir de la distribution des répliques par *bootstrap*.
   (a) Soit $\hat{\mu}_{(.)} = \frac{\Sigma_{b=1}^{R} \hat{\mu}_{(b)}}{R}$.
   (b) Puisque le biais de l'estimateur vaut $Biais_{bs}[\hat{\mu}] = E[\hat{\mu}] - \mu$, l'estimation par *bootstrap* du biais est donné par $Biais_{bs}[\hat{\mu}] = \hat{\mu}_{(.)} - \mu$.
   (c) Puisque $\mu = E[\hat{\mu}] - Biais[\hat{\mu}]$, l'estimation corrigée du biais est donnée par $\hat{\mu}_{bs} = 2\hat{\mu} - \hat{\mu}_{(.)}$.
5. De la même manière, la variance de l'estimation peut être calculée par $Var_{bs}[\hat{\mu}] = \frac{\Sigma_{b=1}^{R} (\hat{\mu}_{(b)} - \hat{\mu}_{(.)})^2}{(R-1)}$.
6. Afin de pouvoir estimer la précision de l'estimateur, l'erreur quadratique de la moyenne, notée *MSE* est donnée par $MSE = Var[\hat{\mu}] + (E[\hat{\mu}] - \mu)^2$. L'unité de la *MSE* est L^2. La racine de la *MSE*, notée *RMSE*, est en L ce qui permet une meilleure appréciation de la précision de l'estimateur.
7. La distribution des densités de probabilité est obtenue à partir des répliques par *bootstrap*. Nous pouvons en déduire l'intervalle de confiance[2] $(100 - \alpha)\%$. L'approximation de cet intervalle est donnée par $[\hat{\mu}_{L,\alpha/2}, \hat{\mu}_{H,\alpha/2}]$ et est appelée la limite de confidence d'Efron.

[1] R est fixé empiriquement à 300. En effet, cela représente un bon compromis entre temps de calcul et précision.
[2] Pour les exemples suivants, α est fixé à dix.

G.2 Résultats

	estimateur	biais	variance	MSE	borne inférieure de l'intervalle de confiance	borne supérieure de l'intervalle de confiance	estimateur corrigé du biais
rotation1	-17.08	0.0628	1.7893	2.0482	-19.14	-14.84	-17.143
rotation2	-13.12	-0.035067	1.4608	1.6795	-15.04	-11.06	-13.085
rotation3	-5.56	-0.091067	1.4402	1.4091	-7.66	-3.82	-5.4689
...	...	...	...	...	...	...	...

TAB. G.1 – Etude de l'estimateur de la moyenne de la différence de luminosité pour chaque rotation. L'intervalle de confidence est calculée avec α valant dix.

En effet, le biais, la variance et donc le *MSE* sont faibles. Afin de mieux représenter l'erreur dans la même unité que les différences étudiées, c'est-à-dire L, le *RMSE* est calculé. L'estimateur calculé présente donc une bonne précision.

G.3 Code Source

compute_estimator.m

```
function [ESTIM_DIFF, estim_distr] = compute_estimator(legomat, alpha, displ, st)
%COMPUTE_ESTIMATOR Summary of this function goes here
%  Detailed explanation goes here
%number of bootstrap samples computed
NBOOT = 300;

%load the DIFF matrix
load(legomat, '-mat');
%compute an estimator for the statistic st, its bias, its variance,
%and its MSE by using the bootstrap method
%create a matrix which contains the st estimator of different samples (matrix
%DIFF) and plot the distribution
for i=1 :1 :size(DIFF,1)  %for each rotation

    ESTIM_DIFF(i,1) = eval([st,'(DIFF(i, :))']); %estimator

    %computation the bootstrap distribution
    bootstat = bootstrp(NBOOT, st, DIFF(i, :));

    ESTIM_DIFF(i,2) = estimator_bias(bootstat, ESTIM_DIFF(i,1)); %bias
    ESTIM_DIFF(i,3) = estimator_var(bootstat);  %variance

    %computation for bootstrap bias corrected estimator
    t = 0;
    for j=1 :1 :length(bootstat)
        t = t + bootstat(j);
    end
    t = t / length(bootstat);
    ESTIM_DIFF(i,7) = 2*ESTIM_DIFF(i,1) - t;  %bootstrap bias corrected estimator

    %confidence interval of estimator
```

```
    [ESTIM_DIFF(i,4), ESTIM_DIFF(i,5)] = conf_inter(bootstat, alpha) ;

    %mean-squared error of the generic estimator
    ESTIM_DIFF(i,6) = ESTIM_DIFF(i,3) + ESTIM_DIFF(i,2)^2 ;
%MSE of the estimator

    %copy the bootstrap distribution for further analysis
    for j=1 :1 :length(bootstat)
        estim_distr(i,j) = bootstat(j) ;
    end
end
%the distribution of estimators for each rotation must be displayed
if displ == 1
    plot(ESTIM_DIFF( :,7), 'r') ;
    xlabel('numero de rotation') ;
    ylabel('estimateur corrige du biais') ;
end
%computation of estimator bias
function [estim_bias] = estimator_bias(V, m)
estim_bias = 0 ;
for i=1 :1 :length(V)
    estim_bias = estim_bias + V(i) ;
end
estim_bias = estim_bias / length(V) ;
estim_bias = estim_bias - m ;
%computation of estimator variance
function [estim_var] = estimator_var(V)
t = 0 ;
for i=1 :1 :length(V)
   t = t + V(i) ;
end
t = t / length(V) ;
estim_var = 0 ;
for i=1 :1 :length(V)
    estim_var = estim_var + (V(i) - t)^2 ;
end
estim_var = estim_var / (length(V)-1) ;
%computation of estimator bootstrap confidence interval (1-alpha)
function [y1, y2] = conf_inter(V, alpha)
%alpha belongs to [0 1]
%there are x = N * alpha/2 points before lower confidence bound
%and idem for upper bound
y = prctile(V, [alpha/2, 100-(alpha/2)]) ;
y1 = y(1) ;
y2 = y(2) ;
```

Annexe H

Acquisition des Données

H.1 Acquisition Aléatoire

H.1.1 Pseudocode

Algorithme 10 Pseudocode de l'algorithme pour l'acquisition de données assistée. Les prises de données sont sauvegardées sous le format spécifié en section 6.3. *nb_ rotations* est fixé à 5. *nb_ itérations* est le nombre de mouvements effectués par le robot. Ce nombre est fourni par l'utilisateur.

Enregistrer les deux mesures de luminosité dans *m1*
FOR $i = 1$ à *nb_ itérations*
  IF $(nb\_itrations\%nb\_rotations) = \left(\frac{i}{nb\_rotations} - 1\right)$ THEN
    //Effectue le déplacement après *nb_ rotations* rotations
    Tirer au hasard une amplitude d'action u_1 et u_2
    Effectuer action $[u_1, u_2]$
    Enregistrer les deux mesures de luminosité dans *lm1*
  ELSE //Effectue la série de rotations
    Tirer au hasard une action u_1
    Effectuer action $[u_1, -u_1]$
    Enregistrer les deux mesures de luminosité dans *lm2*
  ENDIF
  Sauvegarder action $[u_1, -u_1]$, *lm1* et *lm2*
  lm1 = *lm2*
ENDFOR

H.1.2 Code Source

datacquisitional.c

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <time.h>
#include "tropocol.h"
#include "controlutils.h"
int random_method(int i) {
  int temp1, temp2, save1, save2, save3, save4, j, cmd;
  printf("random method :\n");
  if (i<0)
    return -1;
```

```
  FILE* f ;
  f = fopen("data_rotation.csv", "a") ;
  srand(time(0)) ;
  save3 = 0 ;
  save4 = 0 ;
  CONTROL_ROBOT_FUNCTION (&save3, &save4) ;
  for (j=1 ; j<=i ; ++j) {//do i step to take data
    printf("step %i\n", j) ;
    if ((j % ROTATION_NUMBER) == 0) {//do a random movement
      //restrict the set of actions to [-MAX_MOTOR_ACTION, MAX_MOTOR_ACTION]
      temp1 = (int) (MAX_MOTOR_ACTION*3.0*2.0*rand()/(RAND_MAX+1.0)) ;
      temp2 = (int) (MAX_MOTOR_ACTION*3.0*2.0*rand()/(RAND_MAX+1.0)) ;
      temp1 -= 3*MAX_MOTOR_ACTION ;
      temp2 -= 3*MAX_MOTOR_ACTION ;
      save1 = temp1 ;
      save2 = temp2 ;
      cmd = CONTROL_ROBOT_FUNCTION (&temp1, &temp2) ;
      save3 = temp1 ;
      save4 = temp2 ;
    }
    else {//do a random rotation
      //restrict the set of actions to [-MAX_MOTOR_ACTION, MAX_MOTOR_ACTION]
      temp1 = (int) (MAX_MOTOR_ACTION*2.0*rand()/(RAND_MAX+1.0)) ;
      temp1 -= MAX_MOTOR_ACTION ;
      temp2 = -temp1 ;
      save1 = temp1 ;
      save2 = temp2 ;
      cmd = CONTROL_ROBOT_FUNCTION (&temp1, &temp2) ;
      if (cmd == STALL_VALOR)//no stall detected
        fprintf(f, "%i,%i,%i,%i,%i,%i\n", save1, save2, save3, save4, temp1,
temp2) ;//write valors of motors and light sensors in file
      else
        printf("the robot has been stalled ...\n") ;
      save3 = temp1 ;
      save4 = temp2 ;
    }
  }

  fclose(f) ;
  return 0 ;
}
```

H.2 Acquisition Assistée

H.2.1 Pseudocode

Algorithme 11 Pseudocode de l'algorithme pour l'acquisition de données assistée. Les prises de données sont sauvegardées sous le format spécifié en section 6.3. *nb_ rotations* est fixé à 5, ainsi que *b. nb_ itérations* est le nombre de mouvements effectués par le robot. Ce nombre est fourni par l'utilisateur.

Enregistrer les deux mesures de luminosité dans *m1*
FOR $i = 1$ à *nb_ itérations*
  IF $(nb\_itrations\%nb\_rotations) = \left(\frac{i}{nb\_rotations} - 1\right)$ THEN
    //Effectue le déplacement après *nb_ rotations* rotations
    Effectuer action (*rot_ synch*,-*rot_ synch*)
    Effectuer action $[u_1, u_1]$ tel que $u_1 = 5$
    min_ rotation = 2048
    Enregistrer les deux mesures de luminosité dans *lm1*
  ELSE //Effectue la série de rotations
    Tirer au hasard une action u_1
    Effectuer action $[u_1, -u_1]$
    Enregistrer les deux mesures de luminosité dans *lm2*
    IF somme des deux mesures $<$ *min_ rotation* THEN
      min_ rotation = somme des deux mesures
      rot_ synch = 0
    ELSE
      $rot\_synch = rot\_synch + u_1$
    ENDIF
  ENDIF
  Sauvegarder action $[u_1, -u_1]$, *lm1* et *lm2*
  lm1 = *lm2*
ENDFOR

H.2.2 Code Source

datacquisitionass.c

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <time.h>
#include "tropocol.h"
#include "controlutils.h"
#define ROTATION_NUMBER 5
int guided_method(int i) {
  int temp1, temp2, save1, save2, save3, save4, j, cmd, min_l, rot_synch;;
  printf("guided method :\n");
  if (i<0)
    return -1;
  FILE* f;
  f = fopen("data_rotation.csv", "a");
  srand(time(0));
  save3 = 0;
  save4 = 0;
  CONTROL_ROBOT_FUNCTION (&save3, &save4);
  min_l = 2*MAX_LIGHT;
  rot_synch = 0;
```

```
  for (j=1 ; j<=i ; ++j) {//do i step to take data
    printf("step %i\n", j) ;
    if ((j % ROTATION_NUMBER) == 0) {//do a random movement
      min_l = 2*MAX_LIGHT ;
      //back rotation -> try to turn to light
      temp1 = -rot_synch ;
      temp2 = rot_synch ;
      rot_synch = 0 ;
      cmd = CONTROL_ROBOT_FUNCTION (&temp1, &temp2) ;
      //restrict the set of actions to [-MAX_MOTOR_ACTION, MAX_MOTOR_ACTION]
      temp1 = (int) (MAX_MOTOR_ACTION*3.0*2.0*rand()/(RAND_MAX+1.0)) ;
      temp2 = temp1 ;
      temp1 -= 3*MAX_MOTOR_ACTION ;
      temp2 -= 3*MAX_MOTOR_ACTION ;
      save1 = temp1 ;
      save2 = temp2 ;

      cmd = CONTROL_ROBOT_FUNCTION (&temp1, &temp2) ;

      save3 = temp1 ;
      save4 = temp2 ;
    }
    else {//do a random rotation
      //restrict the set of actions to [-MAX_MOTOR_ACTION, MAX_MOTOR_ACTION]
      temp1 = (int) (MAX_MOTOR_ACTION*2.0*rand()/(RAND_MAX+1.0)) ;
      temp1 -= MAX_MOTOR_ACTION ;
      temp2 = -temp1 ;
      save1 = temp1 ;
      save2 = temp2 ;

      cmd = CONTROL_ROBOT_FUNCTION (&temp1, &temp2) ;
      if (cmd == STALL_VALOR)//no stall detected
        fprintf(f, "%i,%i,%i,%i,%i,%i\n", save1, save2, save3, save4, temp1,
temp2) ;//write valors of motors and light sensors in file
      else
        printf("the robot has been stalled ...\n") ;
      save3 = temp1 ;
      save4 = temp2 ;
      if ((temp1+temp2) <= min_l) {
        rot_synch = 0 ;
        min_l = temp1 + temp2 ;
      }
      else
        rot_synch += save1 ;
    }
  }

  fclose(f) ;
  return 0 ;
}
```

Annexe I

Validation de Modèle

I.1 Modèle Linéaire

valid_linear_model.m

```
function [error_distr1, error_distr2] = Valid_linear_model(legomat, display_graph)
%Validation of Lazy Learning model
load(legomat, '-mat');
x = 20;
for i=x :1 :length(X1)
    [error_distr1(i-x+1), error_distr2(i-x+1)] = loo_valid_model(U1(1 :i),
X1(1 :i), X2(1 :i), V1(1 :i), V2(1 :i));
end
if display_graph == 1
    subplot(2,1,1);
    plot(x :1 :length(X1), error_distr1, 'b');
    title('Comparaison entre performance et nombre de donnees utilisees pour
le senseur gauche');
    xlabel('nombre de donnees');
    ylabel('erreur de generalisation');
    subplot(2,1,2);
    plot(x :1 :length(X1), error_distr2, 'r');
    title('Comparaison entre performance et nombre de donnees utilisees pour
le senseur droit');
    xlabel('nombre de donnees');
    ylabel('erreur de generalisation');
end

function [error1, error2] = loo_valid_model(U1, X1, X2, V1, V2)
for i=1 :1 :length(X1)

    %leave-one-out subset
    if i == 1
        ind = [2 :length(X1)];
    else if i == length(X1)
            ind = [1 :length(X1)-1];
        else
            ind = [1 :i-1 i+1 :length(X1)];
        end
    end

        %compute the model for the new subset
```

```
    X = [ones(length(X1)-1,1) U1(ind) X1(ind) X2(ind)] ;
    coef1 = X\V1(ind) ;
    coef2 = X\V2(ind) ;

    V1_h = [1 U1(i) X1(i) X2(i)]*coef1 ;
    V2_h = [1 U1(i) X1(i) X2(i)]*coef2 ;

    %compute the empirical error for the subset
    emp_err1(i) = (V1(i) - V1_h)^2 ;
    emp_err2(i) = (V2(i) - V2_h)^2 ;

end
error1 = mean(emp_err1) ;
error2 = mean(emp_err2) ;
```

I.2 Modèle Non-Linéaire : Lazy Learning

valid_lazy_model.m

```
function [error_distr1, error_distr2] = Valid_lazy_model(legomat, display_graph)
%Validation of Lazy Learning model
load(legomat, '-mat') ;
x = 20 ;
for i=x :1 :length(X1)
    [error_distr1(i-x+1), error_distr2(i-x+1)] = loo_valid_model(U1(1 :i),
U2(1 :i), X1(1 :i), X2(1 :i), V1(1 :i), V2(1 :i)) ;
end
if display_graph == 1
    subplot(2,1,1) ;
    plot(x :1 :length(X1), error_distr1, 'b') ;
    title('Comparaison entre performance et nombre de donnees utilisees pour
le senseur gauche') ;
    xlabel('nombre de donnees') ;
    ylabel('erreur de generalisation') ;
    subplot(2,1,2) ;
    plot(x :1 :length(X1), error_distr2, 'r') ;
    title('Comparaison entre performance et nombre de donnees utilisees pour
le senseur droit') ;
    xlabel('nombre de donnees') ;
    ylabel('erreur de generalisation') ;
end

function [error1, error2] = loo_valid_model(U1, U2, X1, X2, V1, V2)
for i=1 :1 :length(X1)

    %leave-one-out subset
    if i == 1
        ind = [2 :length(X1)] ;
    else if i == length(X1)
            ind = [1 :length(X1)-1] ;
        else
            ind = [1 :i-1 i+1 :length(X1)] ;
        end
    end

    %compute the model for the new subset
    X = [U1(ind) U2(ind) X1(ind) X2(ind)] ;
```

```
    %query points
    Q = [U1(i) U2(i) X1(i) X2(i)] ;
    %matrix of parameters
    id_par = [3 10 ; 3*4 5*4 ; 5*4 8*4] ;

    %scale data
    [sX,mX,vX]=scale(X) ;
    sQ=scale(Q,mX,vX) ;

    V1_h = clqLL(sX, V1(ind), sQ, id_par) ;
    V2_h = clqLL(sX, V2(ind), sQ, id_par) ;

    %compute the empirical error for the subset
    emp_err1(i) = (V1(i) - V1_h)^2 ;
    emp_err2(i) = (V2(i) - V2_h)^2 ;

end
error1 = mean(emp_err1) ;
error2 = mean(emp_err2) ;
```

Annexe J

Politique de Contrôle

J.1 Politique de Contrôle Figée

J.1.1 Pseudocode

Algorithme 12 Pseudocode de l'algorithme pour la politique de contrôle figée. Les actions a et b sont fixées respectivement à 2 et 5, le nombre de rotations pour un tour complet est fixé à 24. La variable *horizon*, fixée à 50, définit le nombre d'itérations effectuées par l'algorithme de contrôle.

FOR *étape* = 1 à *horizon*
  IF *étape* = 1 THEN
    $n\_total = nb\_rotations$
    FOR n =1 à $nb\_rotations$
      Prendre les deux mesures de luminosité *l1* et *l2*
      Enregistrer le minimum *l1*+*l2*
      Sauvegarder n
      Effectuer action (a,-a)
    ENDFOR
  ELSE
    Effectuer action ($-(nb\_rotations/8)$, $(nb\_rotations/8)$)
    $n\_total = nb\_rotations/4$
    FOR n =1 à $nb\_rotations/4$
      Prendre les deux mesures de luminosité *l1* et *l2*
      Enregistrer le minimum *l1*+*l2*
      Sauvegarder n
      Effectuer action (a,-a)
    ENDFOR
  ENDIF
  Prendre le n du minimum des valeurs enregistrées
  Effectuer action $((n\_total - n) * -a, (nb\_total - n) * a)$
  Effectuer action (b,b)
ENDFOR

J.1.2 Code Source

polctrlfigee.c

```
#include <time.h>
int hardwired_control(char* name_file, int horizon, int wm) {
  int i, l1, l2, count_action, tim;
  count_action = 0;
```

```
tim = time(NULL) ;
for (i=0 ; i<horizon ; ++i) {
  int best_rot ;
  printf("step %i\n", i+1) ;
  //repositioning phase
  if (i==0) { //first step -> complete round
    int j, min_l ;
    min_l = 2*MAX_LIGHT ;
    for (j=0 ; j<NBR_CR ; ++j) {
      //store the light measures during a complete round
      if (j == 0) {
        l1 = 0 ;
        l2 = 0 ;
      }
      else {
        l1 = TURN_ACTION ;
        l2 = -TURN_ACTION ;
      }
      CONTROL_ROBOT_FUNCTION (&l1, &l2) ;
      ++count_action ;

      if ((l1+l2) < min_l) {
        best_rot = j ;
        min_l = l1+l2 ;
      }
    }
    //back-rotation
    //printf("best rotation number : %i\n", best_rot) ;
    if (best_rot >= (NBR_CR/2)) {
      l1 = (NBR_CR-best_rot)*(-TURN_ACTION) ;
      l2 = -l1 ;
    }
    else {
      l1 = best_rot*(TURN_ACTION) ;
      l2 = -l1 ;
    }
    CONTROL_ROBOT_FUNCTION (&l1, &l2) ;
    ++count_action ;
  }
  else { //scan in front of the robot
    int j, min_l, turn ;
    turn = (NBR_CR/8) ;
    l1 = -turn*(TURN_ACTION) ;
    l2 = turn*(TURN_ACTION) ;
    CONTROL_ROBOT_FUNCTION (&l1, &l2) ;
    ++count_action ;
    min_l = 2*MAX_LIGHT ;
    for (j=0 ; j<2*turn ; ++j) {
      //store the light measures during a complete round
      if (j == 0) {
        l1 = 0 ;
        l2 = 0 ;
      }
      else {
        l1 = TURN_ACTION ;
        l2 = -TURN_ACTION ;
      }
      CONTROL_ROBOT_FUNCTION (&l1, &l2) ;
```

```
            ++count_action ;
            if ((l1+l2) < min_l) {
              best_rot = j ;
              min_l = l1+l2 ;
            }
          }
          //back-rotation
          l1 = (2*turn-best_rot)*(-TURN_ACTION) ;
          l2 = -l1 ;
          CONTROL_ROBOT_FUNCTION (&l1, &l2) ;
          ++count_action ;
        }
        //closer phase
        l1 = DEFAULT_GO_ACTION ;
        l2 = DEFAULT_GO_ACTION ;
        CONTROL_ROBOT_FUNCTION (&l1, &l2) ;
        ++count_action ;
        printf("\nNumber of actions : %i\nTime of experience (sec) : %i\n\n", count_action, (time(NULL) - tim)
      }
      return 0 ;
    }
```

J.2 Politique de Contrôle Basée sur l'Analyse des Données Sensorielles

J.2.1 Pseudocode

Algorithme 13 Pseudocode de l'algorithme pour la politique de contrôle basée sur l'analyse des données sensorielles utilisant la technique de *bootstrap* afin d'estimer la moyenne de la différence de luminosité. La variable *horizon* est fixée à 50, *nbr_ mesures* à 25, les actions a et b sont fixées respectivement à 2 et 5.

```
FOR index =1 à nbr_ mesures
  Enregistrer les deux mesures de luminosité
ENDFOR
Calculer la différence entre les deux mesures de luminosité
Calculer l'estimateur light1 de la moyenne par la technique de bootstrap
FOR étape = 1 à horizon
  Effectuer action (a,-a)
  IF étape = 1 THEN //Recherche du passage par zéro
    WHILE mesures dans la zone lumineuse THEN
      Effectuer action (2 * a, 2 * −a)
    ENDWHILE
    WHILE position en face de la lumière non trouvée
      FOR index = 1 à nbr_ mesures
        Enregistrer les deux mesures de luminosité
      ENDFOR
      Calculer la différence entre les deux mesures de luminosité
      Calculer l'estimateur light2 de la moyenne par la technique de bootstrap
      IF mesures dans la zone lumineuse THEN
        IF (light2 > 0 AND light1 < 0) OR (light2 < 0 AND light1 > 0) THEN
          //Passage par zéro
          IF |light1| < |light2| THEN
            Effectuer action (-a,a)
          ENDIF
          Position en face de la lumière trouvée
        ENDIF
      ENDIF
      IF position non trouvée THEN
        Effectuer (a,-a)
      ENDIF
      light1 = light2
    ENDWHILE
  ELSE //Recherche du minimum global de la valeur absolue de l'estimateur
    FOR index = 1 à nbr_ mesures
      Enregistrer les deux mesures de luminosité
    ENDFOR
    Calculer la différence entre les deux mesures de luminosité
    Calculer l'estimateur light2 de la moyenne par la technique de bootstrap
    IF |light1| < |light2| THEN
      Action c = -a
    ELSE
      Action c = a
    ENDIF
    light1 = light2
    WHILE |light1| > |light2|
      light1 = light2
      Effectuer (c,-c)
      FOR index = 1 à nbr_ mesures
        Enregistrer les deux mesures de luminosité
      ENDFOR
      Calculer la différence entre les deux mesures de luminosité
      Calculer l'estimateur light2 de la moyenne par la technique de bootstrap
    ENDWHILE
```

J.2.2 Code Source

polctrstat.c

```
#include <time.h>
double mean(int *D, int m) {
  int i ;
  double mean ;
  for (mean=0.0, i=0 ; i<m ; ++i)
    mean += D[i] ;
  mean /= (double) m ;
  return mean ;
}
//computation of the bias corrected estimator
//of light difference by the bootstrap method
double compute_estimator(int *ml, int *mr, int m) {

  int i, j ;
  int *D = (int *) malloc(m*sizeof(int)) ;
  int *Db = (int *) malloc(m*sizeof(int)) ;
  //double *Dst = (double *) malloc(R*sizeof(double)) ;
  double t,  tp ;
  //init the light difference vector
  for (i=0 ; i<m ; ++i)
    D[i] = ml[i] - mr[i] ;
  //init the random generator
  srand(time(NULL)) ;
  //computation of he statistic from the orginal dataset
  t = mean(D, m) ;
  for (tp=0.0, i=0 ; i<B ; ++i) {
    //bootstrap sample
    for (j=0 ; j<m ; ++j) {
      int ind = (int) ((double) m*rand()/(RAND_MAX+1.0)) ;
      Db[j] = D[ind] ;
    }

    //bootstrap estimate distribution
    //Dst[i] = mean(Db, m) ;
    tp += mean(Db, m) ;
  }
  tp /= B ;
  //bootstrap bias corrected
  t = 2.0*t - tp ;
  free(D) ;
  free(Db) ;
  return t ;
}
//computed the mean of the left an the roght sensor to determine the side
int light_info_pos(int *ML, int *MR, int m) {
  int mll, mlr, i ;
  for (mll=0, mlr=0, i=0 ; i<m ; ++i) {
    mll += ML[i] ;
    mlr += MR[i] ;
  }
  mll /= m ;//mean left
  mlr /= m ;//mean right
  return (mll <= LL || mlr <= LL) ? 0 : -1 ;
}
int stat_control(char* name_file, int horizon, int wm) {
```

```
int i, l1, l2, count_action, tim, turn, turn_action ;
double estim, estim2 ;
int* m1 = (int*) malloc(NBR_MEASURES*sizeof(int)) ;
int* m2 = (int*) malloc(NBR_MEASURES*sizeof(int)) ;

count_action = 0 ;
tim = time(NULL) ;
turn_action = TURN_ACTION_STAT ;
for (i=0 ; i<NBR_MEASURES ; ++i) {
  m1[i] = 0 ;
  m2[i] = 0 ;
  CONTROL_ROBOT_FUNCTION (&m1[i], &m2[i]) ;
}

for (i=0 ; i<horizon ; ++i) {
  printf("step %i\n", i+1) ;
  //repositioning phase
  if (i == 0) { //search the pass by 0
    int j ;
    while (light_info_pos(m1, m2, NBR_MEASURES) == 0) {
      //put the robot in the dark side
      printf("position in light side\n") ;
      for (j=0 ; j<NBR_MEASURES ; ++j) {
        if (j == 0) {
          m1[0] = -(4*turn_action) ;
          m2[0] = (4*turn_action) ;
          ++count_action ;
        }
        else {
          m1[j] = 0 ;
          m2[j] = 0 ;
        }
        CONTROL_ROBOT_FUNCTION (&m1[j], &m2[j]) ;
      }
    }
    estim = compute_estimator(m1, m2, NBR_MEASURES) ;

    //the robot is in the dark place -> search the pass by 0
    printf("find pass by 0\n") ;
    turn = 2*turn_action ;
    while (1) { //light direction found ?
      int j ;
      for (j=0 ; j<NBR_MEASURES ; ++j) {
        if (j == 0) {
          m1[0] = turn ;
          m2[0] = -turn ;
          ++count_action ;
        }
        else {
          m1[j] = 0 ;
          m2[j] = 0 ;
        }
        CONTROL_ROBOT_FUNCTION (&m1[j], &m2[j]) ;
      }
      estim2 = compute_estimator(m1, m2, NBR_MEASURES) ;

      printf("estim1 -> estim2 : %g \t-> %g\n", estim, estim2) ;
      if (light_info_pos(m1, m2, NBR_MEASURES) == 0) {
```

```
      printf("position in light side\n") ;
      turn = turn_action ;
      if ((estim > 0 && estim2 < 0) || (estim < 0 && estim2 > 0)) {                    //pass by 0
        if (abs(estim) < abs(estim2)) {
          //back rotation
          l1 = -turn ;
          l2 = turn ;
          CONTROL_ROBOT_FUNCTION (&l1, &l2) ;
          ++count_action ;
        }
        break ; //position found
      }
    }
    estim = estim2 ;
  }
}
else {
  int j, first ;
  turn_action = -TURN_ACTION_STAT ;
  printf("find min\n") ;
  first = 1 ;
  estim2 = compute_estimator(m1, m2, NBR_MEASURES) ;
  do {
    estim = estim2 ;
    for (j=0 ; j<NBR_MEASURES ; ++j) {
      if (j == 0) {
        if (first == 1) {
          m1[0] = turn_action ;
          m2[0] = -turn_action ;
        }
        else {
          m1[0] = -turn_action ;
          m2[0] = turn_action ;
        }
        ++count_action ;
      }
      else {
        m1[j] = 0 ;
        m2[j] = 0 ;
      }
      CONTROL_ROBOT_FUNCTION (&m1[j], &m2[j]) ;
    }
    estim2 = compute_estimator(m1, m2, NBR_MEASURES) ;
    printf("estim1 -> estim2 : %g \t-> %g\n", estim, estim2) ;
    if (abs(estim) <= abs(estim2) && first == 1) {
      //bad direction for rotations
      first = 0 ;
      estim = MAX_LIGHT ;
    }
  }
  while(abs(estim) > abs(estim2))
   ;
  //back rotation
  if (first == 1) {
    l1 = -turn_action ;
    l2 = turn_action ;
  }
  else {
```

```
          l1 = turn_action ;
          l2 = -turn_action ;
        }
        ++count_action ;
        CONTROL_ROBOT_FUNCTION (&l1, &l2) ;
      }
      //closer phase
      l1 = DEFAULT_GO_ACTION ;
      l2 = DEFAULT_GO_ACTION ;
      CONTROL_ROBOT_FUNCTION (&l1, &l2) ;
      ++count_action ;
      printf("\nNumber of actions : %i\nTime of experience (sec) : %i\n\n",
count_action, (time(NULL) - tim)) ;
    }
    free(m1) ;
    free(m2) ;
    return 0 ;
}
```

J.3 Politique de Contrôle Basée sur une Méthode d'Apprentissage

J.3.1 Pseudocode

Algorithme 14 Pseudocode de l'algorithme pour la politique d'apprentissage du contrôle. Les variables *horizon*, b et c sont fixées respectivement à 50, 2 et 5. Le modèle est soit un modèle linéaire, soit le Lazy Learning.

Lire le fichier contenant les données d'apprentissage
Données d'entrée dans X
Données de sortie dans Y
Ajustement du modèle de prédiction à partir de X et Y
Prendre les deux mesures *l1* et *l2* de luminosité pour initialiser l'état du système
FOR *étape* = 1 à *horizon*
  min_ cost = coût maximum
  IF mesures dans la zone lumineuse THEN
    FOR toutes les amplitudes a telles que les actions $(a, -a) \in U$
      Requête de prédiction de *l1_ h* et *l2_ h* à partir du modèle et des entrées *a*, *-a*, *l1* et *l2*
      cost = résultat de la fonction coût à partir de *a*, *-a*, *l1*, *l2*, *l1_ h* et *l2_ h*
      IF *min_ cost* > *cost THEN*
        Enregistrer l'action (*a*,*-a*)
        min_ cost = *cost*
      ENDIF
    ENDFOR
  ELSE //Mesures dans la zone obscure
    IF la dernière action à fait l'objet d'une prédiction THEN
      //Prédiction mauvaise car retour dans la zone obscure
      Enregistrer action (*b*,*-b*) dans la direction inverse de la dernière action
    ELSE //Pas de prédiction pour la dernière action
      Enregistrer action (*b*,*-b*) dans la même direction que l'action précédente
    ENDIF
  ENDIF
  Effectuer action enregistrée
  Prendre les deux mesures de luminosité *l1_ new* et *l2_ new*
  IF le robot n'est pas bloqué dans son mouvement THEN
    Enregistrer les nouvelles données dans le fichier des données d'apprentissage
  ENDIF
  Effectuer action (*c*,*c*) pour avancer le robot
  Prendre les deux mesures de luminosité *l1* et *l2*
ENDFOR

J.3.2 Code Source

J.3.2.1 Modèle Linéaire

polctrllinearmodel.c

```
#include <stdio.h>
#include <gsl/gsl_multifit.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_blas.h>
#include <limits.h>
#include "ll.h"
#include "control.h"
```

```
#include "controlutils.h"
#include <time.h>
int action_distance(int u1, int u2) {
  return ((u1-u2)>=0 ? u1-u2 : u2-u1);
}
double cost_function(int u1, int u2, int x1, int x2, double v1_h, double
v2_h) {
  double cost;
  cost = abs(v1_h-v2_h);
  if (v1_h > LIMIT_LIGHT_DARK && v2_h > LIMIT_LIGHT_DARK)//dark side
    cost += DARK_ADD_VALOR;//to penalize dark side
  return cost;
}
int compute_linear_action(const gsl_matrix* A, const gsl_matrix* B, const gsl_matrix* C, double* x1, doubl
  //linear system : v = A x + B u + C
  //ATTENTION : at the end of the function, x1, x2, u1 and u2 are modified.
  //x1, x2 -> predicted light valors
  //u1, u2 -> best actions
  int i, j;
  gsl_matrix *v, *x, *u, *temp1;
  v = gsl_matrix_alloc(2,1);
  x = gsl_matrix_alloc(2,1);
  u = gsl_matrix_alloc(2,1);
  temp1 = gsl_matrix_alloc(2,1);
  gsl_matrix_set(x, 0, 0, *x1);
  gsl_matrix_set(x, 1, 0, *x2);

  i = u1;
  j = u2;
  gsl_matrix_set(u, 0, 0, i);
  gsl_matrix_set(u, 1, 0, j);
  //use the linear model to estimate the light valors after action
  gsl_blas_dgemm (CblasNoTrans, CblasNoTrans, 1.0, A, x, 0.0, v);
  gsl_blas_dgemm (CblasNoTrans, CblasNoTrans, 1.0, B, u, 0.0, temp1);
  gsl_matrix_add(v, temp1);
  gsl_matrix_add(v, C);
  //the light valors predicted are in matrix v
  //modify input parameters
  *x1 = gsl_matrix_get(v, 0, 0);
  *x2 = gsl_matrix_get(v, 1, 0);
  //free memory allocation
  gsl_matrix_free(v);
  gsl_matrix_free(x);
  gsl_matrix_free(u);
  gsl_matrix_free(temp1);
  return 0;
}
int compute_linear_model(gsl_matrix* X, gsl_vector* v, gsl_vector* c) {
  //from data in matrix X and v, compute, in c, the coefficients of linear
model
  int i;
  double chisq;
  gsl_matrix* cov;
  gsl_multifit_linear_workspace* work;
  if (c->size != X->size2 || X->size1 != v->size) {
    printf("parameter error in compute_linear_model!\n");
    return -1;
  }
```

```
  cov = gsl_matrix_alloc(X->size2, X->size2) ;
  work = gsl_multifit_linear_alloc (X->size1, 5) ;
  gsl_multifit_linear (X, v, c, cov, &chisq, work) ;
  gsl_multifit_linear_free (work) ;
  //if no more needed
  gsl_matrix_free(cov) ;
  //vector c contains the expected coefficients
  return 0 ;
}
int linear_model(const char* file_name, int horizon, int wm, int ad) {
  FILE* f ;
  int i, k, n, x1, x2, u1, u2, y1, y2, count, x1_save, x2_save, last_pred,
count_action, tim ;
  double chisq1, chisq2 ;

  gsl_matrix *X, *cov1, *cov2, *A, *B, *C ;
  gsl_vector *v1, *v2, *c1, *c2 ;
  gsl_multifit_linear_workspace * work ;
  printf("linear model :\n") ;

  n = count_file_lines(file_name) ;
  printf("Initial number of data : %i\n", n) ;
  X = gsl_matrix_alloc (n,5) ;
  v1 = gsl_vector_alloc (n) ;
  v2 = gsl_vector_alloc (n) ;
  c1 = gsl_vector_alloc (5) ;
  c2 = gsl_vector_alloc (5) ;
  cov1 = gsl_matrix_alloc (5,5) ;
  cov2 = gsl_matrix_alloc (5,5) ;
  if ((f = fopen(file_name, "r")) == NULL) {
    printf("error opening file !\n") ;
    return -1 ;
  }
  count_action = 0 ;
  tim = time(NULL) ;
  for (i = 0 ; i < n ; i++) {
    int count = fscanf(f, "%i,%i,%i,%i,%i,%i\n", &u1, &u2, &x1, &x2, &y1,
&y2) ;
    if (count != 6) {
      printf("error reading file\n") ;
      fclose(f) ;
      return -1 ;
    }

    gsl_matrix_set (X, i, 0, x1) ;
    gsl_matrix_set (X, i, 1, x2) ;
    gsl_matrix_set (X, i, 2, u1) ;
    gsl_matrix_set (X, i, 3, u2) ;
    gsl_matrix_set (X, i, 4, 1.0) ;

    gsl_vector_set (v1, i, y1) ;
    gsl_vector_set (v2, i, y2) ;
  }
  fclose(f) ;
#ifdef DEBUG_CONTROL
  printf("\nmatrix X :\n") ;
  print_gsl_mat(X) ;
  printf("\n\nvector v1 :\n") ;
```

```
  print_gsl_vect(v1) ;
  printf("\n\nvector v2 :\n") ;
  print_gsl_vect(v2) ;
  printf("\n") ;
#endif
  compute_linear_model(X, v1, c1) ;
  compute_linear_model(X, v2, c2) ;
  //linear system equation : Y(k+1) = A * X(k) + B * U(k) + C(k)
  A = gsl_matrix_alloc(2,2) ;
  B = gsl_matrix_alloc(2,2) ;
  C = gsl_matrix_alloc(2,1) ;
  //init matrix A
  gsl_matrix_set(A, 0, 0, gsl_vector_get(c1, 0)) ;
  gsl_matrix_set(A, 0, 1, gsl_vector_get(c1, 1)) ;
  gsl_matrix_set(A, 1, 0, gsl_vector_get(c2, 0)) ;
  gsl_matrix_set(A, 1, 1, gsl_vector_get(c2, 1)) ;
  //init matrix B
  gsl_matrix_set(B, 0, 0, gsl_vector_get(c1, 2)) ;
  gsl_matrix_set(B, 0, 1, gsl_vector_get(c1, 3)) ;
  gsl_matrix_set(B, 1, 0, gsl_vector_get(c2, 2)) ;
  gsl_matrix_set(B, 1, 1, gsl_vector_get(c2, 3)) ;
  //init matrix C
  gsl_matrix_set(C, 0, 0, gsl_vector_get(c1, 4)) ;
  gsl_matrix_set(C, 1, 0, gsl_vector_get(c2, 4)) ;
#ifdef DEBUG_CONTROL
  printf("\nmatrix A :\n") ;
  print_gsl_mat(A) ;
  printf("\nmatrix B :\n") ;
  print_gsl_mat(B) ;
  printf("\nmatrix C :\n") ;
  print_gsl_mat(C) ;
  printf("\n") ;
#endif
  //initial state of the robot (x1,x2,0,0))
  x1 = 0 ;
  x2 = 0 ;
  CONTROL_ROBOT_FUNCTION (&x1, &x2) ;
  u1 = 0 ;
  u2 = 0 ;
  if (wm == 1) {
    if ((f = fopen(file_name, "a")) == NULL) {
      printf("error opening file !\n") ;
      return -1 ;
    }
  }
  last_pred = 0 ;
  for (k=0 ; k<horizon ; ++k) {
    int i, j, cost, min_x1, min_x2, min_u1, min_u2, temp, no_pred ;
    double min_cost ;

    printf("\nstep %i :", k) ;
    x1_save = x1 ;
    x2_save = x2 ;
    no_pred = 0 ;
    if (x1>=0 && x1<=1024 && x2>=0 && x2<=1024 && u1>=-MAX_MOTOR_ACTION &&
u1<=MAX_MOTOR_ACTION && u2>=-MAX_MOTOR_ACTION && u2<=MAX_MOTOR_ACTION) {
//no errors in parameters
      min_cost = INT_MAX ;
```

```
      if (x1 <= LIMIT_LIGHT_DARK || x2 <= LIMIT_LIGHT_DARK) {//light side
        last_pred = 0 ;//a prediction will be computed
        for(i=-MAX_MOTOR_ACTION ; i<=MAX_MOTOR_ACTION ; ++i) {
          j = -i ;//rotation action
          if (action_distance(u1, i) <= MAX_CHANGE_ACTION) {
            double v1_h, v2_h ;

            //prediction
            v1_h = x1 ;
            v2_h = x2 ;
            compute_linear_action(A, B, C, &v1_h, &v2_h, u1, u2) ;
#ifdef DEBUG_CONTROL
            printf("\nlight valors predicted by lazy learning model \n
for state (%i,%i) and action (%i,%i) :\n(%g,%g)\n",x1, x2, i, j, v1_h, v2_h ) ;
#endif

            //compute the cost : penalize great valors of light
            //and penalize extreme actions
            cost = COST_FUNCTION_LINEAR (x1, x2, i, j, v1_h, v2_h) ;
            //keep the min cost
            if (cost < min_cost) {
              min_u1 = i ;
              min_u2 = j ;
              min_x1 = v1_h ;
              min_x2 = v2_h ;
              min_cost = cost ;
            }
          }
        }
        printf("\npredicted cost : %g\n", min_cost) ;
      }
      else { //dark side
        if (last_pred == 0) {//last action was in the light side
          last_pred = 1 ;
          no_pred = 1 ;
          if (u1 < u2) {//last motor action is to turn left
            min_u1 = -DEFAULT_TURN_ACTION ;
            min_u2 = DEFAULT_TURN_ACTION ;
          }
          else {
            min_u1 = DEFAULT_TURN_ACTION ;
            min_u2 = -DEFAULT_TURN_ACTION ;
          }
        }
        else {
          //last action was in the dark side -> continue the same turn action
          no_pred = 1 ;
          min_u1 = u1 ;
          min_u2 = u2 ;
        }
      }

    }
    else {
      //errors in parameters -> take another light measures
      //in applying a (0,0) action
      no_pred = 1 ;
      min_u1 = 0 ;
```

```
      min_u2 = 0;
    }
#ifdef DEBUG_CONTROL
    printf("\naction valors choosen for state (%i,%i,%i,%i) : (%i,%i)\n",
x1_save, x2_save, u1, u2, min_u1, min_u2 );
    if (no_pred == 0)
      printf("predicted light valors : (%d,%d)\n", min_x1, min_x2);
#endif
    //update last action valors
    u1 = min_u1;
    u2 = min_u2;
    //action
    x1 = u1;
    x2 = u2;
    temp = CONTROL_ROBOT_FUNCTION (&x1, &x2);
    ++count_action;
    if (temp == STALL_VALOR) {//no stall detected
      if (wm == 1) {//write measures permitted
        //store to increase the dataset
        fprintf(f,"%i,%i,%i,%i,%i,%i\n", u1, u2, x1_save, x2_save, x1, x2);
      }
      if (ad == 1) {//update the model

      }
    }
    else {
      printf("the robot is stalled ...\n");
    }
    printf("real light valors : (%i,%i)\n", x1, x2);
    printf("real cost : %g\n", cost_function(x1_save, x2_save, u1, u2, x1, x2));
    if (no_pred == 0) {//a prediction has been computed -> to delete ?
      //go ahead a little -> second model ?
      x1_save = x1;
      x2_save = x2;
      x1 = DEFAULT_GO_ACTION;
      x2 = DEFAULT_GO_ACTION;

      CONTROL_ROBOT_FUNCTION (&x1, &x2);
      ++count_action;
      printf("\ndefault go action for state (%i,%i) : (%i,%i)\n", x1_save, x2_save, min_u1, min_u2 );
      printf("real light valors : (%d,%d)\n", x1, x2);

    }
    printf("\nNumber of actions : %i\nTime of experience : %i\n\n", count_action, time(NULL)-tim);
  }

  if (wm == 1)
    fclose(f);
  return 0;
}
```

J.3.2.2 Modèle Non-Linéaire : Lazy Learning

polctrllazymodel.c

```
#include <stdio.h>
#include <gsl/gsl_multifit.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_blas.h>
```

```
#include <limits.h>
#include "ll.h"
#include "control.h"
#include "controlutils.h"
#include <time.h>
int action_distance(int u1, int u2) {
  return ((u1-u2)>=0 ? u1-u2 : u2-u1);
}
double cost_function(int u1, int u2, int x1, int x2, double v1_h, double v2_h) {
  double cost;
  cost = abs(v1_h-v2_h);
  if (v1_h > LIMIT_LIGHT_DARK && v2_h > LIMIT_LIGHT_DARK)//dark side
    cost += DARK_ADD_VALOR;//to penalize dark side
  return cost;
}
int lazy_learning_model(const char* file_name, int horizon, int wm, int ad, int forget) {
  //file_name = name of input file
  //wm = write permission -> add new observation in input file
  //ad = adaptive control -> add new observations for control
  //forget = forget factor (e.g. 3 means that the oldest observation
  //is discarded every 3 new observations)
  FILE* f;
  double* X, *V1, *V2;
  int i, k, n, x1, x2, u1, u2, v1, v2, count, x1_save, x2_save, last_pred, count_action, tim, siz;
  gsl_matrix *X_data;
  printf("lazy learning model :\n");
  n = count_file_lines(file_name);
  if (n<40) {
    printf("too small training set!\n");
    return -1;
  }
  if (forget > 0)
    siz = n + horizon * (1 - 1/forget);
  else
    siz = n+horizon;
  V1 = calloc(siz, sizeof(double));
  V2 = calloc(siz, sizeof(double));
  X_data = gsl_matrix_alloc(siz, 4);
  //n+horizon because adaptive control -> add new observations
  //the memory is set to 0 with calloc
  count_action = 0;
  tim = time(NULL);
  printf("Initial number of data : %i\n", n);
  if ((f = fopen(file_name, "r")) == NULL) {
    printf("error opening file!\n");
    return -1;
  }
  i = 0;
  while ((count = fscanf(f, "%i,%i,%i,%i,%i,%i\n", &u1, &u2, &x1, &x2, &v1, &v2)) != EOF) {
#ifdef DEBUG_CONTROL
    //printf("read values : %i %i %i %i %i %i\n", u1, u2, x1, x2, v1, v2);
#endif

    if (count != 6) {
      printf("error reading file!\n");
      fclose(f);
      return -1;
    }
```

```
    //update observations
    gsl_matrix_set(X_data, i, 0, u1);
    gsl_matrix_set(X_data, i, 1, u2);
    gsl_matrix_set(X_data, i, 2, x1);
    gsl_matrix_set(X_data, i, 3, x2);
    V1[i] = v1;
    V2[i] = v2;
    ++i;
  }
  fclose(f);
  X = calloc(n*4, sizeof(double));
  convert_fortran_mat(X, X_data, n);
  //now, X contains the n observations in fortran matrix style
  //initial state of the robot (x1,x2,0,0))
  x1 = 0;
  x2 = 0;
  CONTROL_ROBOT_FUNCTION (&x1, &x2);
  u1 = 0;
  u2 = 0;
  if (wm == 1) {
    if ((f = fopen(file_name, "a")) == NULL) {
      printf("error opening file!\n");
      return -1;
    }
  }
  last_pred = 0;
  for (k=0; k<horizon; ++k) {
    int i, j, cost, min_x1, min_x2, min_u1, min_u2, temp, no_pred;
    double min_cost;

    printf("\nstep %i :", k);
    x1_save = x1;
    x2_save = x2;
    no_pred = 0;
    if (x1>=0 && x1<=1024 && x2>=0 && x2<=1024 && u1>=-MAX_MOTOR_ACTION &&
u1<=MAX_MOTOR_ACTION && u2>=-MAX_MOTOR_ACTION && u2<=MAX_MOTOR_ACTION) {
//no errors in parameters
      min_cost = INT_MAX;
      if (x1 <= LIMIT_LIGHT_DARK || x2 <= LIMIT_LIGHT_DARK) {//light side
        last_pred = 0;//a prediction will be computed
        for(i=-MAX_MOTOR_ACTION; i<=MAX_MOTOR_ACTION; ++i) {
          j = -i;//rotation action
          if (action_distance(u1, i) <= MAX_CHANGE_ACTION) {
            double v1_h, v2_h;
            double Q[4];

            Q[0] = i;
            Q[1] = j;
            Q[2] = x1;
            Q[3] = x2;
            //prediction
            v1_h = ll(n, 4, X, V1, Q);
            v2_h = ll(n, 4, X, V2, Q);
#ifdef DEBUG_CONTROL
            printf("\nlight valors predicted by lazy learning model \n
for state (%i,%i) and action (%i,%i) :\n(%g,%g)\n",x1, x2, i, j, v1_h, v2_h );
#endif
```

```
            //compute the cost : penalize great valors of light and penalize extreme actions
            cost = COST_FUNCTION_LAZY (i, j, x1, x2, v1_h, v2_h) ;
            //keep the min cost
            if (cost < min_cost) {
              min_u1 = i ;
              min_u2 = j ;
              min_x1 = v1_h ;
              min_x2 = v2_h ;
              min_cost = cost ;
            }
          }
        }
        printf("\n\npredicted cost : %g", min_cost) ;
      }
      else { //dark side
        if (last_pred == 0) {//last action was in the light side
          last_pred = 1 ;
          no_pred = 1 ;
          if (u1 < u2) {//last motor action is to turn left
            min_u1 = -DEFAULT_TURN_ACTION ;
            min_u2 = DEFAULT_TURN_ACTION ;
          }
          else {
            min_u1 = DEFAULT_TURN_ACTION ;
            min_u2 = -DEFAULT_TURN_ACTION ;
          }
        }
        else {//last action was in the dark side -> continue the same turn action
          no_pred = 1 ;
          min_u1 = u1 ;
          min_u2 = u2 ;
        }
      }

    }
    else {//errors in parameters -> take another light measures in applying a (0,0) action
      no_pred = 1 ;
      min_u1 = 0 ;
      min_u2 = 0 ;
    }
#ifdef DEBUG_CONTROL
    printf("\naction valors choosen for state (%i,%i) : (%i,%i)\n", x1_save,
x2_save, min_u1, min_u2 ) ;
    if (no_pred == 0)
      printf("predicted light valors : (%d,%d)\n", min_x1, min_x2) ;
#endif
    //update last action valors
    u1 = min_u1 ;
    u2 = min_u2 ;
    //action
    x1 = u1 ;
    x2 = u2 ;
    temp = CONTROL_ROBOT_FUNCTION (&x1, &x2) ;
    ++count_action ;
    if (temp == STALL_VALOR) {//no stall detected
      if (wm == 1) {//write measures permitted
        //store to increase the dataset
        fprintf(f,"%i,%i,%i,%i,%i,%i\n", u1, u2, x1_save, x2_save, x1, x2) ;
```

```
        }
        if (ad == 1) {//update the model
          //add new observations for lazy learning
          gsl_matrix_set(X_data, n, 0, u1);
          gsl_matrix_set(X_data, n, 1, u2);
          gsl_matrix_set(X_data, n, 2, x1_save);
          gsl_matrix_set(X_data, n, 3, x1_save);

          //recreate fortran matrix X
          X = realloc(X, (n+1)*4*sizeof(double));
          convert_fortran_mat(X, X_data, n+1);

          V1[n] = x1;
          V2[n] = x2;

          ++n;//increment number of observations
          printf("update dataset -> %i data\n", n);
          if (forget > 0 && k != 0 && (k%forget) == 0) { //discard the oldest observation
            //modify the matrix X_data, V1 and V2 by a shift
            int j;
            printf("forget a data -> %i data\n");
            --n;
            for (j=0; j<n; ++j) {
              gsl_matrix_set(X_data, j, 0, gsl_matrix_get(X_data, j+1, 0));
              gsl_matrix_set(X_data, j, 1, gsl_matrix_get(X_data, j+1, 1));
              gsl_matrix_set(X_data, j, 2, gsl_matrix_get(X_data, j+1, 2));
              gsl_matrix_set(X_data, j, 3, gsl_matrix_get(X_data, j+1, 3));
              V1[j] = V1[j+1];
              V2[j] = V2[j+1];
            }
          }
        }
      }
      else {
        printf("the robot is stalled ...\n");
      }
      printf("real light valors : (%i,%i)\n", x1, x2);
      printf("real cost : %g\n", cost_function(x1_save, x2_save, u1, u2, x1, x2));
      if (no_pred == 0) {//a prediction has been computed -> to delete ?
        //go ahead a little -> second model ?
        x1_save = x1;
        x2_save = x2;
        x1 = DEFAULT_GO_ACTION;
        x2 = DEFAULT_GO_ACTION;

        CONTROL_ROBOT_FUNCTION (&x1, &x2);
        ++count_action;
        printf("\ndefault go action for state (%i,%i) : (%i,%i)\n", x1_save,
x2_save, min_u1, min_u2 );
        printf("real light valors : (%d,%d)\n", x1, x2);

      }
      printf("\nNumber of actions : %i\nTime of experience : %i\n\n",
count_action, time(NULL)-tim);
    }

    if (wm == 1)
      fclose(f);
```

```
  //free memory allocation
  gsl_matrix_free(X_data) ;
  free(X) ;
  free(V1) ;
  free(V2) ;
  return 0 ;
}
```

Bibliographie

[1] P. Andrews and J. Stuber. Lejos. http ://lejos.sourceforge.net/, 2003.

[2] P. Andrews and J. Stuber. lejos documentation. http ://lejos.sourceforge.net/ api-docs/index.html, 2003.

[3] P. J. Antsaklis. Neural network in control systems. *IEEE Control Systems Magazine*, 10(3-5), 1990.

[4] R. C. Arkin. *Behavior-Based Robotics*. MIT Press, 1998.

[5] M. Ash. Lnp - lego network protocol. http ://legos.sourceforge.net/HOWTO/x405.html, 2000.

[6] K. J. Astrom and B. Wittenmark. *E. Cliffs*. Prentice Hall, 1990.

[7] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning for control. *Artificial Intelligence Review*, 11(1-5) :75–113, 1997.

[8] Efron B. Bootstrap methods : another look at the jacknife. *The Annals of Statistics*, 7(1-26), 1979.

[9] Brian Bagnall. *Core LEGO Mindstorms Programming*. Prentice Hall PTR, 2002. Unleash the Power of the Java Platform.

[10] S. Baluja. Evolution of an artificial neural network based autonomous land vehicle controller. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 26(3) :450 – 463, June 1996.

[11] D. J. Barnes. Teaching introductory java through lego mindstorms models. Technical report, University of Canterbury, 2002.

[12] P. Batavia, D. Pomerleau, and C. Thorpe. Applying advanced learning algorithms to alvinn. Technical Report CMU-RI-TR-96-31, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, October 1996.

[13] D. Baum. Not quite c. http ://www.baumfamily.org/nqc/, 2003.

[14] S. Bergbreiter and K. Pister. Cotsbots. http ://www-bsac.eecs.berkeley.edu/ sbergbre/CotsBots/cotsbots.html, 2001.

[15] M. Birattari and G. Bontempi. The lazy learning toolbox. Technical report, Université Libre de Bruxelles, 1999.

[16] G. Bontempi. *Local Learning Techniques for Modeling, Prediction and Control*. PhD thesis, Université Libre de Bruxelles, 1998.

[17] G. Bontempi. *Cours de Modèles Stochastiques II*. Université Libre de Bruxelles, 2003.

[18] G. Bontempi, M. Birattari, and H. Bersini. Lazy learning for modeling and control design. *International Journal of Control*, 72(7/8) :643–658, 1999.

[19] G. Bontempi, M. Birattari, and H. Bersini. A model selection approach for local learning. *Artificial Intelligence Communications*, 13(1) :41–48, 2000.

[20] G. Bontempi, M. Birattari, and H. Bersini. The local paradigm for modeling and control : from neuro-fuzzy to lazy learning. *Fuzzy Sets and Systems*, 121(1) :59–72, 2001.

[21] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 1986.

[22] R. Brooks. A robot that walks : Emergent behavior from a carefully evolved network. *Neural Computation*, 1989.

[23] R. Brooks. The behavior language : User's guide. *MIT AI Lab MEMO 1127*, 1990.

[24] R. Brooks. New approaches to robotics. *Science*, 253, 1991.

[25] W. Cleveland and C. Loader. Smoothing by local regression : Principles and methods. *Computational Statistics*, 1995.

[26] W.S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, 1979.

[27] LEGO Company. Lego mindstorms official page. http ://mindstorms.lego.com, 1997.

[28] M. Cornelius. Lnpd. http ://legos.sourceforge.net/files/linux/LNPD/, 2002.

[29] Sony Corporation. Aibo. http ://www.aibo-europe.com/, 2001.

[30] C. Croarkin. Nist/sematech e-handbook of statistical methods. http ://www.itl.nist.gov/ div898/handbook/, 2003.

[31] J. Dalbey. Pseudocode standard. http ://www.csc.calpoly.edu/ jdalbey/SWE/pdl_std.html, 2003.

[32] Université Libre de Bruxelles. Printemps des sciences. http ://www.ulb.ac.be/ infors-ciences/inforsc_3.html, 2003.

[33] M. Dorigo. Alecsys and the autonomouse : Learning to control a real robot by distributed classifier systems. *Machine Leanring*, 19(3) :209–240, 1995.

[34] M. Dorigo and M. Colombetti. Robot shaping : Developping autonomous agents through learning. *Artificial Intelligence*, 71(2) :321–370, 1994.

[35] M. Dorigo and U. Schnepf. Genetics-based machine learning and behaviour based robotics : A new synthesis. *IEEE Transactions on systems*, 23(1) :141–154, 1993.

[36] N. R. Draper and H. Smith. *Applied Regression Analysis*. John Wiley and Sons, 1981.

[37] J.J. Droesbeke. *Eléments de Statistques*. 1988.

[38] R. M. Barnett et al. Review of particle properties. *Physical Review*, 1996. http ://pdg.lbl.gov/.

[39] H. Friedl and E. Stampfer. Resampling methods. Technical report, Institute of Statistics, Technical University Graz, 2001.

[40] O. Fuentes, R. Rao, and M. Van Wie. Hierarchical learning of reactive behaviors in an autonomous mobile robot. In *Fuentes, O., R.P.N. Rao, and M. Van Wie, Hierarchical learning of reactive behaviors in an autonomous mobile robot, Proc., IEEE Int'l. Conf. on Systems, Man and Cybernetics, Vancouver, BC, Canada, October 1995.*, 1995. citeseer.nj.nec.com/fuentes95hierarchical.html.

[41] P. Gaudiano, E. Zalama, and J. Coronado. An unsupervised neural network for low-level control of a wheeled mobile robot : Noise resistance. *IEEE Trans. Systems, Man, Cybernetics*, 1996.

[42] P. Gaussier and A. Revel. Living in a partially structured environment : How to bypass the limitations of classical reinforcement techniques. Technical report, ETIS ENSEA, 1996.

[43] P. Good. *Permutation Tests - A Practical Guide to Resampling Methods for Testing Hypotheses*. Springer-Varlag New York, Inc., 2000.

[44] H. Gross. A neural architecture for sensorimotor anticipation. *Generative Character of Percept*, (12) :1101–1129, 1999.

[45] R. Hempel. pbforth. http ://www.hempeldesigngroup.com/lego/pbForth/homePage.html, 2002.

[46] HITACHI. *H8/300 Programming Manual*, 1990.

[47] HITACHI. *Hitachi Single-Chip Microcomputer H8/3297 Series Hardware Manual*, 1990.

[48] D. W. Hogg, F. Martin, and M. Resnick. Braitenberg creatures. Technical report, MIT Media Laboratory, 1991. ftp ://cherupakha.media.mit.edu/pub/el-publications/EL-Memos/memo13.PS.Z.

[49] HONDA. Humanoid robot p3. http ://www.honda-robots.com/english/html/p3/ frameset2.html, 2001.

[50] The MathWorks Inc. Matlab. Version 6.5.0.180913a Release 13, 2002.

[51] Stanford Research Institue. Shakey. http ://www.sri.com/about/timeline/shakey.html, 1969.

[52] Deep Ocean Exploration Institute. Sentry. http ://www.whoi.edu/institutes/doei/general/mission.htm, 2003.

[53] IRIDIA, IDSIA, EPFL, and CNR-IP. Swarm-bots. http ://www.swarm-bots.org/, 2001.

[54] T. Jochem, D. Pomerleau, and C. Thorpe. Maniac : A next generation neurally based autonomous road follower. In *Proceedings of the International Conference on Intelligent Autonomous Systems*, February 1993. Also appears in the Proceedings of the Image Understanding Workshop, April 1993, Washington D.C., USA.

[55] K. A. Johansen. Light-weight distributed shared memory for small mobile commodity robots. Master's thesis, Université de TromsÃ¸, 2002.

[56] P. K. Khosla and C. J. J. Paredis. Millibots. http ://www-2.cs.cmu.edu/ cyberscout/millibots.html, 1999.

[57] J.J. Leonard and H.F. Durrant-Whyte. Mobile robot localization by tracking : Geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3) :376–382, 1991.

[58] LogIT. Logit sensors for lego rcx. http ://www.dcpmicro.com/lego/, 2000.

[59] H. H. Lund and L. Pagliarini. Robot soccer with lego mindstorms. *Lecture Notes in Computer Science*, 1604 :141, 1999.

[60] Fred G. Martin. Crickets. http ://lcs.www.media.mit.edu/people/fredm/projects/cricket/, 2002.

[61] L. Massone. Sensorimotor learning. In *The Handbook of Brain Theory and Neural Networks*. MIT Press, 1994.

[62] Sun microsystems. Java. http ://java.sun.com/.

[63] A. W. Moore. *Efficient Memory-based Learning for Robot Control*. PhD thesis, University of Cambridge, 1990.

[64] C. H. Moore. Forth. http ://www.forth.com/, 1970.

[65] NASA. Mars pathfinder. http ://mars.jpl.nasa.gov/mer/newsroom/pressreleases/20030709a.html, 1996.

[66] St. Nelson. Introduction to the legos kernel. 2000.

[67] M. L. Noga. Brickos. http ://brickos.sourceforge.net/, 2003.

[68] M. L. Noga. brickos documentation. http ://brickos.sourceforge.net/docs/APIs/html-c/, 2003.

[69] S. J. Olszanskyj, J. M. Lebak, and A. Bojanczyk. Modification methods for recursive least squares problems. *Numer. Algorithms*, 7 :325–354, 1994.

[70] N. Patil. Mindsensors. http ://www.mindsensors.com/, 2000.

[71] N. Patil. Mindsensors - active multipexor. http ://www.mindsensors.com/active_mux.htm, 2000.

[72] M. Patrick. Legos programming - drive control library. http ://www.mastincrosbie.com/mark/legOS/motorlib.html, 2002.

[73] D. Pomerleau. Neural network vision for robot driving. In M. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. 1995.

[74] K. Proudfoot. Rcx internals. http ://graphics.stanford.edu/ kekoa/rcx/, 1998.

[75] M. H. Quenouille. Approximate tests of correlation in time-series. *Journal of the Royal Statistical Society*, 1949.

[76] O. Rensfelt and D. Rosen. http ://www.docs.uu.se/docs/research/projects/selnet/lunar/ulunar/, 2003.

[77] D. E. Rumelhart and D. Zipser. *Parallel Distributed Processing : Explorations in the Microstructure of cognition, Volume 1 : Foundations*. MIT Press, 1986.

[78] S. Russel and P. Norvig. *Artificial Intelligence : A Modern Approach*. Prentice Hall, 1995.

[79] K-TEAM SA. Khepera. http ://www.k-team.com/robots/khepera/index.html, 2003.

[80] A. Saffiotti. The uses of fuzzy logic in autonomous robot navigation : a catalogue raisonné. Technical report, Orebbro University, 1997.

[81] F. Samuelsson. Distributed evolution of behaviour for a group of social autonomous agents. http ://www.dtek.chalmers.se/ d4sama/Kurser/Exjobb/report/report.html, 2001.

[82] S. Schaal and C. G. Atkeson. Robot juggling : An implementation of memory-based learning. *Control Systems Magazine*, 14(1) :57–71, 1994.

[83] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, (36) :111–147, 1974.

[84] E. D. Taillard. A statistical test for comparing success rate. In *MIC2003 : The Fifth Metaheuristics International Conference*, 2003.

[85] S. H. G. ten Hagen, D. l'Ecluse, and B. J. A. Krose. Q-learning for mobile robot control. In E. Postma and M. Gyssens, editors, *BNAIC'99, Proc. of the 11th Belgium-Netherlands Conference on Artificial Intelligence*, pages 203–210, 1999.

[86] S. Thrun, J. Schulte, and C. Rosenberg. Interaction with mobile robots in public places. Technical report, University of Pittsburg, 2000.

[87] P. D. Turney. A theory of cross-validation error. *Journal of Experimental and Theoretical Artificial Intelligence*, 6 :361–391, 1994.

[88] P. Wira, J.-P. Urban, and J. Gresser. Filtrage de kalman à modèle multiple et à dynamique adaptative : Application à la poursuite de cible par asservissement visuel robotique. Technical report, Université de Haute-Alsace, 2002.