CHAPITRE 1

REVUE DE LITTÉRATURE

1.1 Introduction

Cette section a pour but de présenter le contexte de la programmation des contrôleurs robotiques utilisés dans le cadre de la recherche appliquée. Premièrement, une mise en contexte de la recherche appliquée en robotique est effectuée. Deuxièmement, les différentes méthodes de programmation des robots sont exposées. Troisièmement, les différents types de contrôleurs disponibles sont décrits afin de mettre en relief quels sont les types de contrôleurs les plus appropriés dans un contexte de recherche appliquée.

1.2 Profil actuel de la robotique industrielle

Les robots industriels prennent de plus en plus de place dans l'industrie. Entre 1986 et 2002, le parc de robot installés aux États-Unis a plus que quadruplé [6]. Selon la Commission Économique de l'Europe des Nations Unies (UNECE) et de la Fédération Internationale de Robotique (IFR), le nombre de commandes de robots industriels, tout pays confondus, a augmenté de 93.9 % en 2004 par rapport à 1996 [7]. Ces informations montrent que la robotique industrielle est en plein essor, et ce, depuis un bon moment.

Plusieurs facteurs contribuent à rendre ce moyen de production davantage prépondérant. Selon Colestock [8], l'accroissement de la popularité des robots industriels est due à l'augmentation de leur précision et de leur puissance mécanique. De plus, l'auteur soutient que cet accroissement de performance a été conjugué à une baisse des coûts des robots de l'ordre de 70% par rapport à 1995: un robot de 100 000$ en 1995 se vendrait 30 000$ en 2005 [8].

À ce jour, les robots industriels fabriqués par les grands manufacturiers sont principalement utilisés pour faire de l'assemblage, de la soudure ou de l'emballage et de la palettisation [9]. D'autres applications existent, notamment dans l'industrie alimentaire et botanique, mais elles demeurent marginales.

Ces limites d'application peuvent toutefois être repoussées grâce à la recherche appliquée en robotique, qui est présentée à la section suivante.

1.3 La recherche en robotique industrielle

Les grands manufacturiers ne sont pas en mesure de combler tous les besoins de l'industrie, comme le prouvent les chiffres qui suivent. Selon l'Associated Press [10], l'investissement annuel en recherche robotique serait de 100 millions de dollars américains pour l'Europe, du même montant pour la Corée et le Japon. D'après le même article [10], les États-unis investiraient 500 millions annuellement pour la recherche en robotique.

Toutefois, selon Dutkiewicz et coll.[11], les chercheurs qui travaillent avec des robots industriels ont à faire face à de nombreux problèmes dus au fait que les contrôleurs de robots industriels sont fermés et non standardisés. Les auteurs [11] soutiennent que pour être en mesure d'avoir un robot plus adapté pour leurs expérimentations, ils doivent soit construire un nouveau manipulateur ainsi qu'un contrôleur ou encore reprogrammer un contrôleur existant au complet.

Développer ou reprogrammer un contrôleur de robot demande beaucoup d'efforts car les systèmes mécaniques, électriques, électroniques et informatiques doivent être agencés harmonieusement. Ces systèmes sont brièvement présentés dans la section qui suit.

1.4 Anatomie d'un système robotique industriel

Tout système robotique, bien qu'il en existe une grande variété, possède certaines composantes standard pouvant être catégorisées. Une catégorisation de ces composants est tentée dans cette section de façon à mettre en relief la place qu'occupe le contrôleur de robot dans ce type de système.

La configuration générale d'un système robotique générique est donc la suivante [12]:

1. Le mécanisme du robot;
2. Les capteurs externes;
3. Le contrôleur de robot;
4. Le système de simulation et de programmation hors-ligne.

Il est pertinent de décrire les composantes de ce système, dont l'interaction est illustrée à la Figure 2. Le mécanisme du robot est caractérisé par ses composantes physiques (membrures, actionneurs, capteurs, etc.) qui accomplissent concrètement les tâches. Les capteurs externes permettent d'effectuer des mesures de l'environnement pour permettre au programme de prendre des décisions quant aux actions à exécuter.

Le contrôleur de robot orchestre le fonctionnement du mécanisme d'après les informations des capteurs et de sa programmation. Ce dernier fait donc le lien entre les utilisateurs, les capteurs et les actionneurs. Le système de simulation permet à l'utilisateur de voir le comportement programmé du robot avant d'exécuter le programme sur le système physique, ce qui évite d'éventuels bris. Le système de programmation hors-ligne permet quant à lui de programmer une tâche avant de l'exécuter sur le robot. Il est important de préciser qu'il est maintenant possible que le système de programmation hors-ligne et le simulateur soit à l'intérieur du contrôleur.

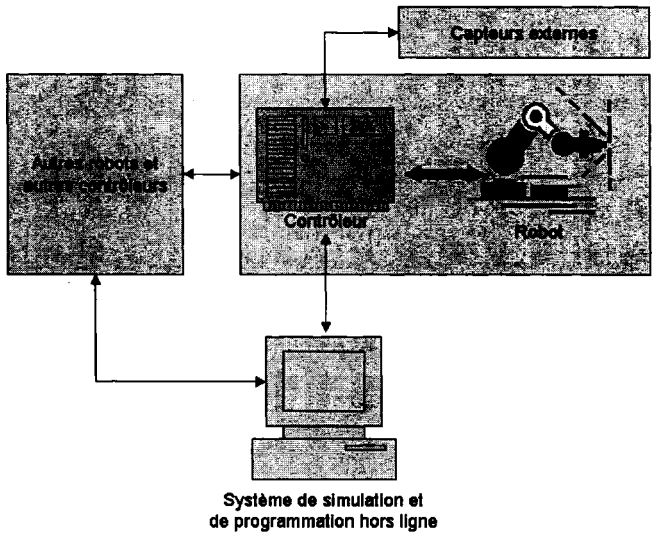

Figure 2 Illustration de la configuration générale d'un système robotique [12]

Le cœur du système est le contrôleur de robot car c'est par lui que transite l'ensemble des informations. Le développement de ce dernier nécessite de nombreuses heures de développement car il s'agit d'une tâche critique et complexe. Selon Loffler, Dawson et coll. [13], la difficulté de ce travail réside particulièrement dans le fait qu'il nécessite une expertise dans plusieurs domaines, notamment en robotique, en programmation temps réel, en intégration de matériel et en gestion d'accès simultanés.

Pour ajouter à la complexité, plusieurs types de programmation sont possibles à l'intérieur même du contrôleur. La distinction entre ces différents types de programmation est présentée dans la sous-section suivante.

1.5 L'organisation logicielle d'un contrôleur

D'après Kapoor et Tesar [14], un contrôleur peut être divisé en trois couches logicielles distinctes (voir Figure 3), soit la couche d'interface matérielle, la couche opérationnelle et la couche de programmation robotique.

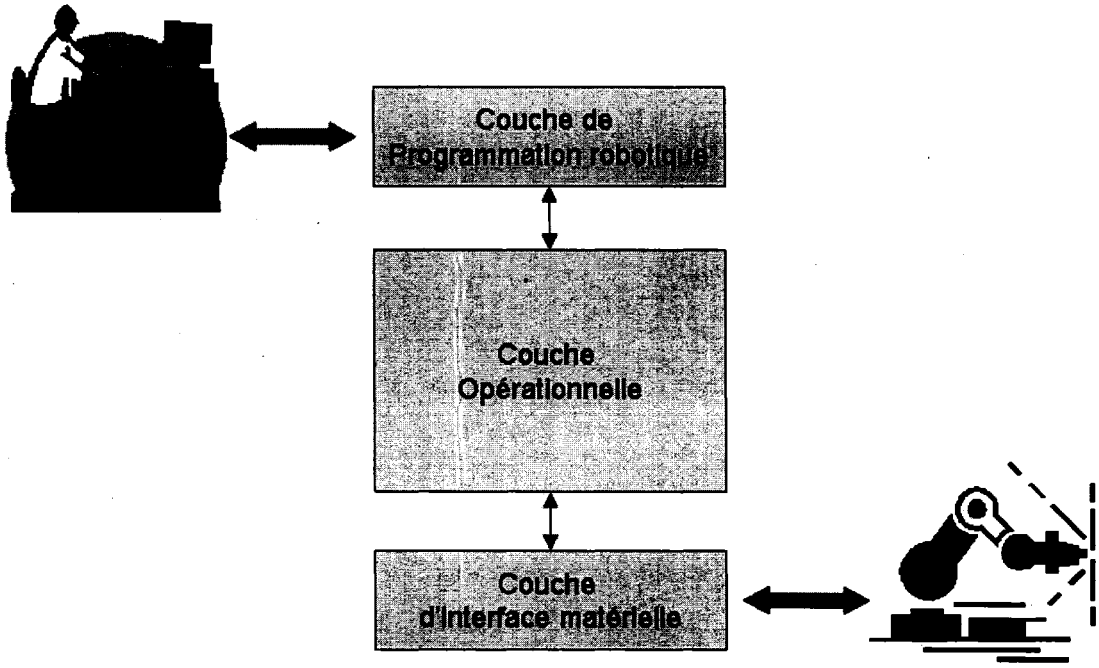

Figure 3 Illustration de l'organisation logicielle d'un contrôleur de robot

La couche d'interface matérielle a pour but de servir d'interface entre les logiciels de servocommande, les périphériques et les bus de communication [14]. Du point de vue pratique, il s'agit du système d'exploitation et des pilotes de périphériques.

La couche de programmation robotique est l'interface homme-machine du système robotique [14]. Deux types de programmation sont possibles dans cette couche, soit la programmation manuelle et la programmation assistée. Ils se définissent comme suit [15]:

- Programmation manuelle: programmation, typiquement sans le robot, qui se fait hors ligne. Elle peut s'effectuer graphiquement avec un schéma, un diagramme de fluence (*flowchart*) ou un graphe. Cette programmation peut également se faire textuellement dans un langage soit:

  - Propre au manufacturier du contrôleur, par exemple avec Karel, Rapid, etc.;
  - Générique, par exemple avec C++, Java, etc.;
  - Comportemental, par exemple avec Yampa [16], dérivé du langage Haskell [17].

- Programmation assistée: programmation typiquement effectuée lorsque le robot est en ligne (*on-line*), c'est-à-dire que sa programmation nécessite que le robot soit sous tension. Trois méthodes catégorisent ce type de programmation:
  - Par l'exemple: programmation assistée la plus courante, utilisant généralement un boîtier de commande (*teach pendant*) et plus rarement la voix ou le geste. L'usager fait bouger le manipulateur, le contrôleur enregistre les actions effectuées (mouvement, préhension, etc.) dans le but de les reproduire sur demande.
  - Par l'information: programmation basée sur les informations que le robot reçoit via un de ses capteurs, qui a pour effet de déclencher l'exécution d'une tâche connue a priori ou d'un modèle d'action généralement prédéfini par programmation.
  - Par l'apprentissage: programmation rarement utilisée se basant sur le renforcement ou la répression de certains comportements (réseau de neurones, etc.).

La couche opérationnelle fait le lien entre les demandes des clients, les informations reçues des capteurs, des contraintes de sécurité et de tout ce qui n'est pas directement lié à l'usager ou au matériel. Cette couche assure la communication entre les couches d'interface matérielle et de programmation, supportant donc le mode de programmation manuelle ou assistée ou les deux, en plus de gérer l'accès au matériel, de gérer les communications, etc.: c'est le point central du contrôleur de robot.

Cette catégorisation des couches logicielles montre que la couche de programmation, comprenant la programmation manuelle et assistée, est davantage liée à l'exécution d'une tâche précise du robot tandis que la couche opérationnelle permet de faire fonctionner le système: sans elle, rien n'est possible.

La partie logicielle qui sera abordée dans ce mémoire concernera uniquement le développement de la couche opérationnelle. Avant d'aborder le développement de cette couche, il convient de présenter au sein de quels types de contrôleur elle s'effectue.

1.6 Types de contrôleurs robotiques

La couche opérationnelle diffère selon le type de contrôleur utilisé. Cette section présente les différents types de contrôleurs documentés dans la littérature. Selon Dixon, Moses et coll.[18], il existe quatre types de contrôleur robotique:

1. contrôleur propriétaire;
2. contrôleur développé avec un langage de programmation traditionnel;
3. contrôleur développé avec un cadre d'application robotique;
4. contrôleur développé grâce à un environnement de simulation graphique avec un générateur de code.

Ces différents types seront décrits dans les quatre sous-sections qui suivent. Le but de cet exercice est de bien comprendre quels sont les contrôleurs qui sont le plus appropriés pour la recherche appliquée en robotique.

1.6.1 Le contrôleur propriétaire

La première alternative, le contrôleur fourni par un fabricant, permet de programmer facilement les tâches à effectuer par le robot: c'est un système clé en main. Dans ce cas,

le fabricant fournit un contrôleur de robot complet et fonctionnel avec un interpréteur de commande qui est spécifique au langage propriétaire.

Les commandes disponibles sont celles qui sont liées à la définition et à l'exécution des tâches pour laquelle le robot a été conçu. Elles incluent généralement le mouvement entre deux points, l'approche normale vers une surface, la préhension ou le dépôt d'un objet, etc. Ce langage textuel contient les principales commandes robotiques et ressemble souvent à du basic, comme le montre la portion de code Karel suivante:

```
$MOTYPE = JOINT          -- activer le mode articulaire
MOVE TO depart           -- aller à la position départ
OPEN HAND 1              -- ouvrir le préhenseur 1
MOVE TO distributeur     -- aller chercher la pièce
CLOSE HAND 1             -- agripper la pièce
MOVE TO boite            -- aller à la position de la boîte
OPEN HAND 1              -- déposer la pièce
```

Il existe trois types de contrôleurs propriétaires [19]:

1. Fermé: système où il est très difficile, voire impossible, de modifier le matériel et la structure logicielle pour quelqu'un d'autre que le manufacturier original. Ce sont les contrôleurs les plus fréquemment rencontrés dans l'industrie. Ils possèdent comme avantage d'être économiques et éprouvés. C'est la solution la plus intéressante lorsque le contrôleur répond aux exigences de l'application désirée;
2. Ouvert: systèmes qui permettent de modifier tous les aspects, qu'ils soient logiciels ou matériels. Comme ces contrôleurs sont généralement des produits uniques, ils sont plus onéreux et demandent plus de validation que les contrôleurs propriétaires;
3. Hybride: système semi-ouvert qui permet d'accéder de l'extérieur aux fonctions de bas niveau. Des compagnies ont mis en marché dans les années 1990 ces contrôleurs qui tentent d'allier les avantages des contrôleurs fermés et ouverts. Par exemple, cette approche permet de connecter un ordinateur standard sur un

contrôleur hybride qui peut aller jusqu'à servir uniquement de boîte d'amplification pour les moteurs. L'auteur souligne que cette approche a réduit les coûts de systèmes robotiques car elle permet la réutilisation tant de matériel éprouvés que de logiciels fiables. La compagnie allemande Kuka produit un tel type de contrôleur [20].

Presque chaque compagnie robotique d'envergure possède son propre langage textuel dont quelques exemples sont donnés au Tableau I.

Tableau I

Les fabricants de robots et leurs langages associés

Compagnie	**Langage**
Fanuc / GMF	Karel
CRS	Rapl
KUKA	KRL
ABB	Rapid
Adept	V+

Un langage de programmation propriétaire doit être utilisé en conjonction avec un contrôleur compatible, habituellement fourni par le fabricant à l'achat du manipulateur. Ce langage est habituellement facile à utiliser mais a comme principal désavantage d'être peu adaptable à une nouvelle tâche pour laquelle le système robotique n'a pas été conçu. Cette spécialisation pour la robotique, bien qu'elle apporte de la simplicité au utilisateurs des systèmes robotiques, peut parfois être limitative: il peut être difficile de modifier la boucle de contrôle ou encore de créer un protocole de communication sur mesure. De plus, ce type de langage ne peut être utilisé que lorsqu'il existe un contrôleur compatible avec le robot à utiliser, ce qui limite encore davantage les possibilités.

1.6.2 Le contrôleur basé sur un langage de programmation traditionnel

La seconde alternative consiste à développer un contrôleur au complet avec un langage traditionnel, tel le C++, le Pascal, le Java, etc. Cette approche est extrêmement flexible mais demande énormément de temps de développement ainsi qu'une connaissance très poussée des contrôleurs de robots en général, ce qui n'est pas à la portée de tous. De plus, chaque nouveau contrôleur est reprogrammé à partir de zéro, ce qui génère une perte de temps considérable.

1.6.3 Le contrôleur développé avec un cadre d'application robotique

La troisième alternative est le contrôleur développé à l'aide d'un Cadre d'Application Robotique (CAR). C'est en fait un cadre d'application de type « Infrastructure système » adapté aux contraintes et réalités des systèmes robotiques.

Le Grand dictionnaire terminologique de l'Office de la langue française [21] définit un cadre d'applications (appelé en anglais *application framework* ou encore *object-oriented application framework*) de la façon suivante:

« Infrastructure logicielle qui facilite la conception des applications par l'utilisation de bibliothèques de classes ou de générateurs de programmes en programmation orientée objet. »

Il existe plusieurs cadres orientés objet, dont la *Microsoft Foundation Class* (MFC), les *Distributed Component Object Model* (DCOM) et les applets Java. Selon Fayad et Schmidt [22], un cadre d'application orienté objet a les avantages suivants:

- Modularité: encapsule les détails d'implémentation pour faciliter la compréhension et la maintenance du logiciel;

- Réutilisation: définit des composants génériques pouvant être réemployés pour créer de nouvelles applications;
- Extensibilité: permet d'étendre une interface stable et son comportement à une autre interface qui requiert un comportement similaire;
- Inversion du contrôle: appel du code de l'usager par le cadre d'application plutôt que l'inverse.

Par contre, ces mêmes auteurs [22] mentionnent que ce type d'approche technologique comporte également ses désavantages:

- Les connaissances requises pour développer un cadre d'applications appartiennent souvent à un nombre limité d'experts;
- Apprendre à utiliser un cadre d'applications orienté objet de façon efficace requiert des efforts considérables;
- Il peut être difficile d'intégrer plusieurs cadres d'application car ils n'ont pas nécessairement été conçus pour cela;
- Effectuer la maintenance d'un cadre d'application requiert une compréhension profonde de ce dernier;
- Le déverminage et la validation d'un programme développé à l'aide d'un cadre d'application peuvent être difficiles pour plusieurs raisons:
    - Les bogues peuvent venir de plusieurs sources, notamment du cadre d'application, de l'utilisation faite de ce dernier ou encore du code du programme développé à l'aide du cadre d'application;
    - L'inversion de contrôle peut rendre plus difficile la localisation d'un bogue;
    - Le code du cadre d'application peut ne pas être disponible;
    - Le code du cadre d'application peut être mal compris des usagers.

Malgré la liste des désavantages, les auteurs [22] soutiennent que les cadres d'applications orientés objet seront le noyau de l'avant-garde technologique en matière

de développement logiciel au cours du 21ième siècle. Il est intéressant de constater que cet avis se reflète dans les publications du IEEE [23]: une recherche du mot clé *framework* par année, toutes disciplines confondues, montre qu'il y a une augmentation constante du nombre de documents publiés depuis les 15 dernières années (voir Figure 4).

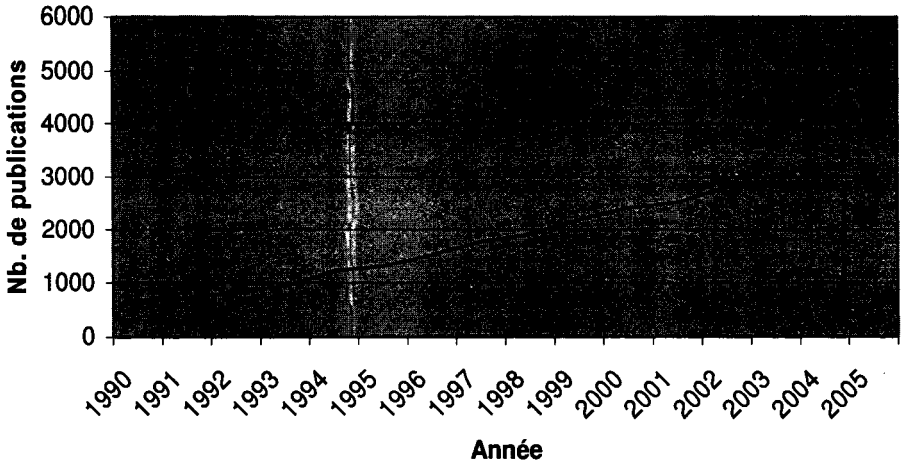

Figure 4 Évolution du nombre de publications par année de l'IEEE traitant de « framework » [23]

Un Cadre d'Application Robotique (CAR) est simplement l'application à la robotique des cadres d'application exposés précédemment. Un CAR est une bibliothèque modulaire de fonction couplée à une structure d'exécution appliquée à la robotique et développée dans un langage traditionnel. Il permet d'avoir la grande flexibilité du développement à l'aide d'un langage traditionnel tout en diminuant considérablement le temps de développement.

Plusieurs CAR existent, dont Orocos[24], Oscar [25] et Microb [26]. Ces CAR ont été développés dans différents contextes mais toujours en réponse au manque de contrôleurs de robots ouverts et flexibles disponibles sur le marché.

Orocos est l'acronyme de « Open RObot COntrol Software ». Ce projet à code source ouvert subventionné par la Communauté Économique Européenne (CEE) était à l'origine composé de quatre institutions de recherche: l'Université Catholique de Leuven [27], le KTH en Suède [28], le LAAS en France [29] et l'ULM en Allemagne [30]. Son développement a été coordonné de la Belgique par le docteur Herman Bruyninckx. L'objectif fondateur de ce CAR est de fournir des outils de développement de contrôleurs standardisés et adaptés à la recherche académique ainsi qu'à l'éducation [31].

Toutefois, l'union de plusieurs centres de recherche ne s'est pas avérée fructueuse car l'intégration logicielle des segments développés dans les différentes universités n'a jamais eu lieu [32]. La portion française, le « LAAS Open Software for Autonomous Systems » [29], ne semble plus disponible publiquement. La contribution allemande, le « OROCOS::SmartSoft » [33], est distribué séparément tout comme la contribution suédoise « ORCA » [32]. Ces deux contributions sont davantage des boîtes à outils modulaires que de véritables CAR. Un seul cadre d'application distinct a finalement émergé de cette union: OROCOS. OROCOS est encore activement développé, maintenu, utilisé et distribué sous licence LGPL [34] à ce jour par la portion belge du projet [24].

Oscar est l'acronyme de « Operational Software Components for Advanced Robotics » et a été développé à l'Université du Texas à Austin par le Robotic Research Group [35]. Ce projet, fondé sur la thèse du docteur Kapoor [36], a deux principales orientations: prodiguer des outils de commande optimale et favoriser la reconfiguration pour les robots redondants [14]. Malgré que l'auteur parle de l'évolution du CAR via la contribution de ses utilisateurs [14], le code source ainsi que les binaires ne sont pas disponibles et les détails sur la licence sont introuvables sur le site officiel . Mis à part la présence des publications Robotic Research Group [35], rien n'indique que le CAR est encore en évolution.

Microb est l'acronyme de « Module Intégré de Contrôle ROBotique » et a été développé par l'Institut de Recherche d'Hydro-Québec (IREQ). Le démarrage de ce projet a principalement été motivé par les objectifs suivants (extraits de [37], p.6):

- Concevoir une bibliothèque commerciale de manière à garantir la robustesse des différents robots utilisés « ... »;
- Garantir l'évolution continue de la bibliothèque pour éviter d'avoir à reconstruire tous les contrôleurs « ... »;
- Faciliter la gestion des ressources « ... » en s'assurant que Microb est un produit indépendant de ses concepteurs;
- Créer un outil de pointe permettant à Hydro-Québec de conserver son leadership mondial en matière de contrôle robotique;
- Créer de nouveaux projets d'innovation « ... ».

Microb est utilisé dans le cadre de plusieurs projets robotiques (voir Figure 5) à Hydro-Québec, notamment pour un manipulateur [38], un robot sous-marin [39] ou encore comme une boîte à outil pour un logiciel de planification de meulage robotisé [26]. Deux projets étudiants utilisent actuellement Microb pour développer leurs contrôleurs: le club Walking Machine de l'ÉTS [40] (robot mobile) et Groupe Robofoot ÉPM [41] (multi-robots). Auparavant, d'autres projets universitaires ont déjà utilisé cette technologie (Hélicoptère [26], SARCOS [26], SONIA-ÉTS [42] et SAE Robotique [43]) mais rien n'indique qu'ils utilisent encore Microb.

Le projet a été entrepris peu avant 2000 et sa distribution sous licence *Lesser General Public License* (LGPL) [34] a débuté en 2002. L'utilisation de cette licence avait pour but de permettre d'élargir le nombre d'utilisateurs pour dynamiser le développement et d'assurer la pérennité du CAR [37]. Malheureusement, malgré le fait que Microb soit encore distribué à ce jour, il semble qu'il soit davantage utilisé à l'IREQ qu'ailleurs, ce qui oblige l'IREQ à supporter les coûts de maintenance. De plus, comme l'évolution de

Microb dépend de ses utilisateurs et que ces derniers sont peu nombreux, elle est relativement lente.

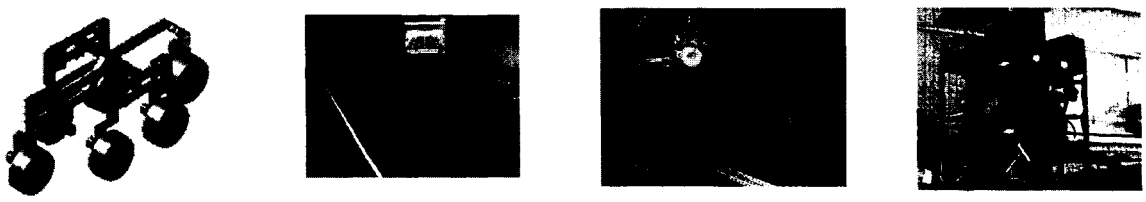

Figure 5 Plateformes robotisées utilisant Microb
(source: [3])

Après analyse de ces différents CAR, il apparaît que les caractéristiques communes sont généralement les suivantes:

- Polyvalent: conçu pour tous types de robots;
- Flexible: s'adapte aux besoins spéciaux en évoluant;
- Modulaire: permet de sélectionner un ou plusieurs modules, selon les besoins;
- Portable: généralement utilisable sous plusieurs systèmes d'exploitation;
- Indépendant: non lié spécifiquement à un manufacturier de robot ou de matériel;
- Standard: structure toujours les contrôleurs de la même façon;
- Réutilisable: diminue le temps de développement via la réutilisation du code;
- Adaptable: exécuter le même code en simulation que lors de l'exécution temps réel;
- Convenable: contient des outils de base liés à la robotique (cinématique, dynamique, génération de trajectoire, etc.).

Toutefois, un CAR demande à l'usager d'avoir des connaissances avancées en programmation dans un langage orienté objet et des systèmes temps réels. De plus, cette approche a tous les désavantages liés aux cadres d'application (voir ci-dessus).

1.6.4 Le contrôleur développé en prototypage rapide

Le quatrième choix, la génération de code via un outil de simulation graphique tel Matlab/Simulink ou Scilab/Scicos, est associée à la notion de prototypage rapide de contrôleurs (PRC), appelée en anglais *rapid controller prototyping* (RCP). Cette approche modifie le processus traditionnel de conception de contrôleurs en automatisant les étapes de développement logiciel.

La conception est vue comme un processus itératif, qui nécessite plusieurs itérations avant d'aboutir au produit répondant aux spécifications. Budgen [44] présente la conception comme étant un problème malicieux (*wicked*). Il donne quelques caractéristiques d'un problème malicieux:

- Il n'y a pas de bonnes ou de mauvaises solutions au problème, seulement quelque chose entre les deux;
- Il n'y a pas de formulation définitive ni de fin au problème;
- Il n'y a pas de test immédiat ou ultime pour évaluer la solution;
- Il n'y a pas un nombre de solution défini au problème;
- Il s'agit d'un problème unique.

La conception n'est pas un processus linéaire comme celui de la résolution d'un problème scientifique [44]. Elle est souvent perçue comme un cheminement suivant une spirale dont le point le plus près du centre représente l'obtention d'une solution jugée acceptable, comme le montre la Figure 6.

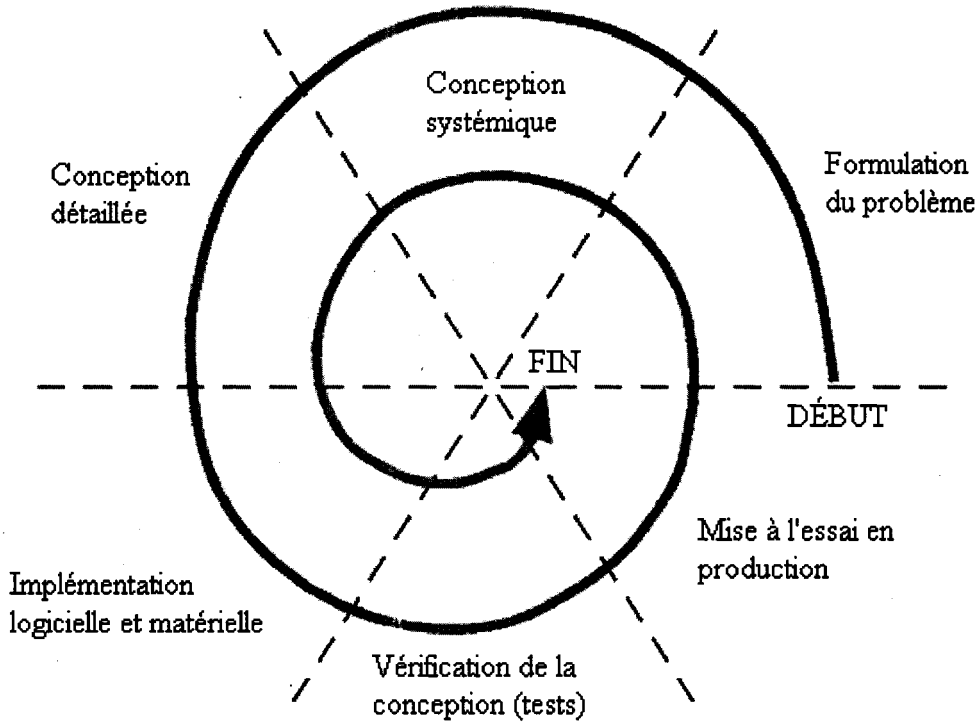

Figure 6 Illustration du processus de design [45]

Comme présenté à la Figure 7, deux principaux facteurs différencient le processus de prototypage rapide:

1. L'itération est brève et automatique dans le cas du prototypage rapide au lieu d'être manuelle. La brièveté de l'itération est due au fait qu'elle est gérée par un seul environnement de développement conçu à cette fin. Cette approche permet de se rapprocher plus rapidement du centre de la spirale de conception;
2. La partie logicielle et matérielle est déjà prise en charge par le logiciel, ce qui retire une étape du processus. Cela permet au concepteur de mettre l'accent sur les algorithmes plutôt que de mettre du temps sur le design informatique du contrôleur.

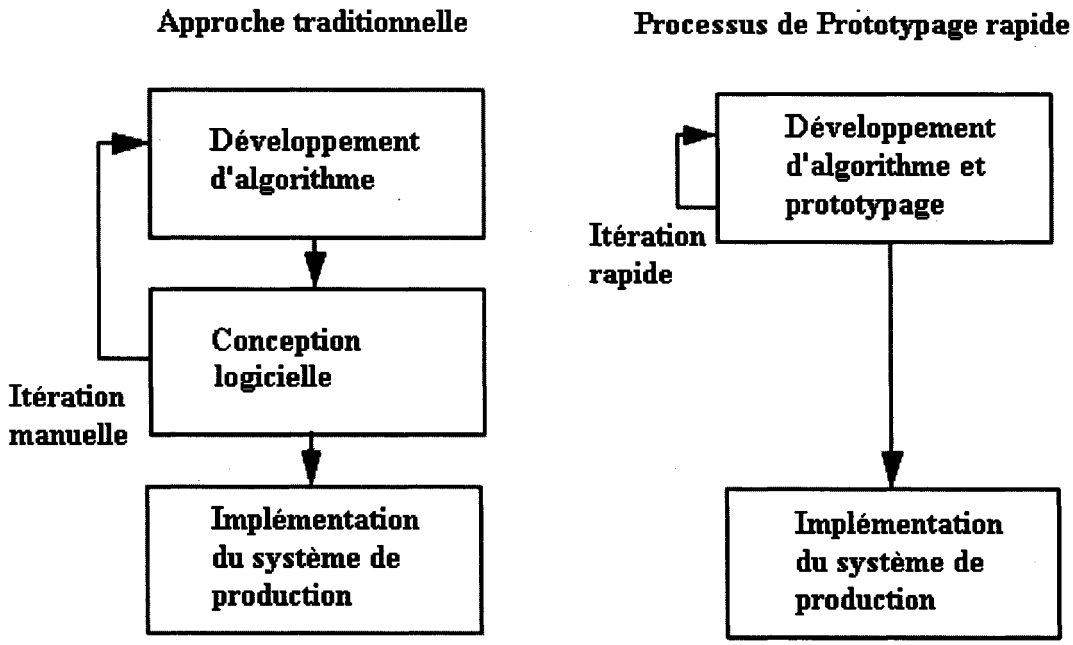

Figure 7 Comparaison graphique entre l'approche traditionnelle et le prototypage rapide [45]

Le PRC permet donc de rapidement simuler et expérimenter des algorithmes de contrôle sans avoir à consacrer beaucoup d'efforts pour développer la plateforme logicielle. Ces environnements de simulation graphique possèdent de puissants outils de calcul et de modélisation particulièrement adaptés au contrôle ainsi qu'à la simulation de système réels.

Trois environnements de simulation munis de générateur de code existent dans la littérature, soit la suite Matlab / Simulink / RTW [2], la suite Scilab /Scicos [46] et la suite NI MATRIXx / SystemBuild / Autocode [47]. En se basant sur le nombre d'articles disponibles sur l'engin de recherche d'IEEE [23], il est possible de déduire que les produits de Mathworks sont davantage utilisés que la suite Scilab / Scicos. En effet, une recherche sur cet engin des termes « Scilab et Scicos » donne seulement 2 résultats tandis qu'une recherche de « Matlab et Simulink » retourne 601 documents différents. Il est à noter que « Matrixx et SystemBuild » ne retourne aucun document.

La première suite, l'environnement Matlab / Simulink est développé par la compagnie Mathworks et possède un générateur de code nommé Real-Time Workshop (RTW)[45]. Les produits de cette compagnie sont une référence dans le domaine de l'ingénierie: le logiciel Matlab est disponible depuis 1984 et le générateur de code existe depuis 1994 [45]. Le générateur de code possède de nombreuses plateformes cibles sur lesquelles le code peut être exécuté, que ce soit sur des systèmes d'exploitation temps réels que sur des microcontrôleurs. Ce générateur peut être personnalisé via la modification de fichiers en format texte responsables de la génération du code ou de sa compilation.

La suite Matlab / Simulink / RTW est un outil très utilisé en recherche, en éducation et en industrie. Les applications robotiques utilisant le PRC sont très présentes dans la littérature: certains l'utilisent pour concevoir un contrôleur pour un robot parallèle [48], d'autre pour un robot à 6 degrés de liberté avec vision artificielle [49] ou encore pour remettre à jour un contrôleur de robot désuet, en l'occurrence le Puma 560, pour l'utiliser à des fins de recherche [18] ou d'éducation [50]. De façon générale, le PRC est appliqué à des systèmes extrêmement diversifiés (automobile[51], moteurs [52], convoyeur [53], etc.). La compagnie Mathworks présente sur son site Internet [2] plus de 54 différentes organisations (compagnies, centres de recherches, etc.) qui ont utilisés leur suite logicielle avec succès.

La seconde suite, Scilab / Scicos, est un logiciel à code source libre plus récent que la suite de Mathworks: son développement a débuté en 1990 et aucune mention de la génération de code n'a été fait avant 2001 dans la littérature [54]. Cet environnement s'intègre harmonieusement avec RTAI-Lab [55], module qui génère du code destiné à être exécuté sur la plateforme temps réel Linux RTAI. L'acronyme RTAI vient de l'anglais *Real-Time Application Interface*. Il est à noter que le générateur de code est également compatible avec les fichiers Simulink mais que la seule cible disponible est la plateforme temps réel Linux RTAI.

La suite Scilab / Scicos est utilisée comme outil de PRC principalement en recherche, plus particulièrement en Suisse et en Italie. Les applications présentées dans la littérature ne sont pas appliquées à la robotique mais plutôt à des applications simples, par exemple le contrôle d'un moteur [56] ou d'un pendule [57]. Quelques écoles américaines utilisent Scilab dans un but éducatif [58; 59] mais uniquement en simulation logicielle.

La troisième suite, soit Matrix-x / System build, est développé par la compagnie National Instruments et possède un générateur de code nommé Autocode [60] qui peut soit produire du code C ou Ada. Le matériel informatique compatible est limité au matériel supporté par LabVIEW Real-Time [61] ou conforme avec le standard PXI [62].

Par contre, le PRC n'est pas complètement flexible car il ne supporte pas tous les systèmes d'exploitation, ni tout le matériel informatique. Il a été nécessaire dans plusieurs cas de développer des pilotes compatibles avec la plateforme visée [49; 53].

De façon générale, il est difficile d'implanter des mécanismes de contrôle de haut niveau liés à la sécurité des opérations de systèmes réels tels les superviseurs ou chiens de garde (*watchdog*). Le problème avec les outils de PRC réside dans le fait que ces derniers sont davantage conçus pour effectuer du traitement de signal que pour faire de la gestion événementielle.

1.7 Les contrôleurs de robots dans un cadre de recherche appliquée

Les chercheurs ont besoin de plateformes robotisées flexibles pour permettre d'expérimenter de nouvelles avenues. De plus, il arrive fréquemment que ces plateformes soient conçues sur mesure tant au niveau mécanique, électrique et électronique. Sachant cela, il convient d'évaluer les alternatives afin de déterminer lesquelles sont les plus appropriées pour la recherche appliquée.

La première option, l'utilisation du contrôleur propriétaire (voir 1.6.1), ne peut être envisagée que lorsqu'un fabricant est en mesure de fournir un contrôleur compatible avec le système robotique à développer. De plus, comme les chercheurs ont besoin d'une plateforme possédant une grande flexibilité et qu'il n'y a que peu de contrôleurs commerciaux qui permettent d'avoir un contrôle logiciel total, cette solution est à proscrire dans la majorité des cas. Toutefois, s'il arrive qu'un contrôleur de type ouvert soit totalement compatible avec le système robotique et qu'il soit en mesure de répondre au cahier de charges sans aucune modification dans sa conception, cette solution peut être retenue.

La seconde option, qui consiste à développer un contrôleur à partir de zéro à l'aide d'un langage traditionnel (voir 1.6.2), est contre-indiquée pour la recherche car elle entraîne une perte de temps trop importante par rapport aux avantages qu'elle peut apporter en terme de flexibilité.

La troisième option, soit le développement de contrôleur en utilisant un cadre d'application robotique (voir 1.6.3), est très intéressante pour la recherche. Elle permet d'allier la flexibilité de la seconde méthode tout en réduisant drastiquement le temps de développement total. Par contre, en plus des difficultés inhérentes à l'utilisation d'un cadre d'application standard, l'utilisation du CAR demande des connaissances avancées en informatique. Malgré le fait que cette solution soit très viable, elle peut ne pas être la solution idéale pour un spécialiste en contrôle dont la compétence première n'est pas la programmation.

La quatrième option, le contrôleur développé à l'aide d'un environnement de simulation graphique avec un générateur de code (voir 1.6.4), est très attirante pour le chercheur qui ne désire pas se lancer dans une longue et coûteuse activité de programmation car il peut obtenir rapidement des résultats concrets avec le matériel. Cette approche a, par contre, le désavantage d'être moins flexible que la précédente: le matériel supporté peut s'avérer

limité et les mécanismes de supervision, pour la sécurité par exemple, peuvent s'avérer plus difficiles à implanter. De plus, comme le code est généré automatiquement, il peut être difficile de le modifier manuellement, ce qui est parfois requis avec un système robotique utilisé en recherche appliquée.

1.8 Conclusion

À la lumière de ces informations, il est dégagé que la première et la seconde option sont peu appropriées pour la recherche appliquée dans la plupart des cas. Le problème de la première option est son manque de flexibilité tandis que la seconde option demande beaucoup trop de ressources. Les deux dernières options sont de bonnes avenues pour la recherche appliquée et sont utilisées tant par les instituts de recherche que par les universités.

À ce jour, il ne semble pas exister d'approche pour profiter à la fois des avantages du PRC et du CAR. Une approche combinant les deux options pourrait potentiellement accélérer le développement des contrôleurs tout en garantissant à ces derniers un maximum de flexibilité.

Dans le chapitre 2, les deux approches qui semblent appropriées pour la recherche appliquée seront mises en contexte au sein de deux organismes dans lesquels elles sont employées (section 2.2). Cette mise en contexte aboutira à la présentation de la problématique (section 2.3), puis à la proposition d'une solution (sections 2.4 à 2.7) alliant deux approches de développement de contrôleur.