Environnements logiciels pour la perception et l'action en robotique

Eric Marchand
INRIA, IRISA, projet Lagadic
Campus de Beaulieu, Rennes
Eric.Marchand@irisa.fr

Résumé - ***Si le nombre de plates-formes robotiques est en constante évolution, la mise en œuvre logicielle du cycle perception/action reste un problème délicat. En 15 ans, les progrès rapides des technologies matérielles ont cependant permis de passer de logiciels ad-hoc s'exécutant sur des machines ou des cartes spécialisées au déploiement d'algorithmes de perception et de commande complexes sur des plates-formes matérielles peu coûteuses et non spécialisées. Il existe malheureusement relativement peu de plates-formes logicielles "génériques" permettant de les commander et encore moins d'outils permettant d'assurer le lien entre les processus de perception et la commande.***
Nous verrons qu'il est cependant possible de mettre en avant quelques éléments permettant de résoudre ces difficultés et de présenter des architectures logicielles permettant partiellement d'assurer un certain caractère générique.
Nous illustrerons cet article avec quelques boites à outils logicielles disponibles le plus souvent en open-source, permettant d'assurer ce lien entre la perception et la commande des robots.

I. Introduction

Les différents articles présentés dans les actes de cette conférence montrent bien la complexité et l'hétérogénéité des systèmes que nous sommes amenés à considérer. Cette complexité se retrouve tout aussi bien au niveau du matériel (mécatronique) à concevoir, des tâches à spécifier, des techniques de commande ou de perception que de l'architecture logicielle à mettre en œuvre.

Les systèmes robotiques auxquels nous nous intéresserons ici sont des systèmes intrinsèquement réactifs. En effet, ils doivent uniquement répondre aux sollicitations du monde extérieur et aucun processus de planification n'entre en jeu dans leur conception (ce n'est évidemment pas le cas de tous les systèmes robotiques). Ce sont de plus des systèmes temps-réel. En effet, les informations sur lesquelles est fondé leur comportement sont extraites de l'environnement (par exemple des images acquises par une caméra). Si ces systèmes étaient des systèmes passifs, le concept de système temps-réel n'entrerait pas en jeu, mais nous nous plaçons dans un contexte perception/action et, de ce fait, les informations fournies par les capteurs doivent être traitées et analysées afin de donner une nouvelle commande au robot et ce, avant qu'une nouvelle information ne soit acquise. Ainsi, le concept de temps-réel intervient à ce niveau : les informations doivent être traitées "*le plus rapidement possible*", mais, surtout, elles doivent l'être avant que l'information suivante ne soit disponible. Si l'on admet qu'une telle classification des différents systèmes a un sens, les systèmes réactifs appartiennent, la plupart du temps, à la catégorie des systèmes temps-réel. Ils ont un certain nombre de spécificités et leur mise en œuvre doit (ou devrait) être à même de répondre à certains critères.

Des architectures logicielles plus ou moins complexes ont donc été proposées depuis plusieurs années pour gérer de manière plus ou moins convaincante les liens entre le bas niveau (commandes, perception et mettant le plus souvent en œuvre des algorithmes "continus" et le haut niveau gérant les objectifs globaux (mettant en œuvre des algorithmes discrets reposant principalement sur la manipulation d'informations symboliques). Parmi ces architectures, on retrouvera les architectures centralisées, les architectures hiérarchiques, les architectures comportementales et, plus récemment, des architectures hybrides (où l'accent est mis sur la décomposition entre les couches décisionnelles et fonctionnelles) [1], [25], [3].

Les architectures robotiques sont par définition des systèmes critiques où les aspects de sécurité sont cruciaux. Dans [24], l'auteur se focalisait principalement sur les aspects de sécurité de fonctionnement logiciel. De plus en plus de systèmes font maintenant appel aux méthodes formelles pour valider les spécifications et la conformité de l'implémentation aux spécifications. Dans certains cas, un code "valide" peut même être automatiquement généré.

Finalement un dernier objectif important est de pouvoir appréhender, dans une même architecture, l'hétérogénéité matérielle des systèmes considérés. Dans le contexte d'une diffusion de l'architecture logicielle hors du laboratoire source, en fonction du contexte cible et donc des tâches à réaliser, l'architecture matérielle, les systèmes de capteurs et l'architecture informatique cible, du système peuvent être très différents et cette hétérogénéité doit être prise en compte dès la conception de l'architecture logicielle. C'est le cas par exemple du système CLARAty [25], Orccad [3], et dans une certaine mesure ViSP [22].

Dans la section II, nous donnerons quelques détails sur certaines techniques permettant d'implémenter certaines des caractéristiques que nous venons d'évoquer. Dans la section III nous évoquerons le problème du prototypage rapide en prenant

pour exemple les système d'asservissement visuel.

II. Des systèmes réactifs et hétérogènes

A. Contraintes issues des systèmes réactifs

Les systèmes robotiques tels que nous les considérons ici sont donc des systèmes réactifs [16]. Un tel système peut être divisé en deux parties : l'environnement et un second sous-système qui en est la partie automatisée. Le comportement de cette partie du système (le robot dans la plupart des cas) est directement dicté par les informations acquises dans l'environnement. La conséquence d'une telle définition est que ces systèmes doivent être déterministes (pour une même suite de signaux en entrée, ils doivent fournir une même suite de signaux de sortie), ils sont soumis à des contraintes temporelles (fréquence de modification de l'environnement ou d'acquisition des capteurs), ils doivent supporter le parallélisme (arrivée simultanée d'informations), et finalement, ils doivent être sûrs (optimalement, on devrait pouvoir prouver que leur comportement est cohérent vis-à-vis des spécifications).

Ces systèmes peuvent donc généralement être divisés en trois niveaux : un niveau "continu" (cycle perception action élémentaire) et un niveau "discret" (niveau tâche ou mission) et éventuellement un niveau de planification[1]. L'une des difficultés inhérente à la mise en œuvre de ces systèmes vient du fait que les techniques et les modèles correspondants à la gestion des systèmes échantillonnés/continus sont très différents des techniques et des modèles nécessaires au contrôle de tâche (automates, systèmes à évènements discrets).

B. Environnements de mise en œuvre

La plupart des environnements disponibles reposent sur l'utilisation de langages impératifs conventionnels (C, C++ ou même Matlab) et ne permettent pas, de façon simple, d'assurer les propriétés de concurrence ou de sûreté que nous avons évoquées dans la section II. Dans un contexte de prototypage rapide (voir section III), ils permettent cependant un développement rapide et aisé (car reposant sur des bibliothèques de fonctions spécialisées importantes), le plus souvent multi plates-formes et permettent (presque) toujours de réaliser des simulations. Le recours aux OS temps-réels permet de prendre en compte un certain nombre des contraintes que nous venons d'évoquer. L'utilisation de langages de plus haut niveaux, de langages fonctionnels (eg, Functional Reactive Programming) voire synchrones permet, tout en améliorant le potentiel d'abstraction, d'assurer un gain dans le processus de fiabilisation du code généré. Ces techniques restent cependant, à l'heure actuelle, peu utilisées. Des environnements dédiés peuvent aussi être considérés, apportant un gain notable tant sur les aspects de sûreté que sur les facilités de mise en œuvre.

1) Langages de "haut niveau": Comme nous l'avons vu, il serait souhaitable d'utiliser un formalisme permettant un certain degré de parallélisme. Le modèle le plus courant relatif au parallélisme est asynchrone (on retrouve là les langages de type Ada,Occam et les systèmes d'exploitation (OS) temps-réel). Les langages comme Occam (CSP) ou Ada ont de nombreuses qualités. Ils sont bien structurés et permettent en général un bonne modularité. Cependant, reposant sur une hypothèse asynchrone, ils ne sont pas déterministes. La synchronisation entre les processus est réalisée à l'exécution et est de fait non prédictible. La seconde approche repose sur la connexion de programmes classiques (écris par exemple en C/C++) en utilisant les primitives de synchronisation d'un noyau temps-réel (RTAI, RT Linux, Vx Works,...). La principale difficulté vient ici du nombre de programmes à analyser et à connecter : la conception, le diagnostic et la maintenance sont relativement complexes. Les contraintes temporelles ne sont pas exprimées dans les programmes mais sont assurées en utilisant les primitives de synchronisation et de communication de l'OS. Ceci amène généralement à des systèmes non déterministes, sur lesquels aucune propriété de sécurité, vivacité, ou autres, ne peut être vérifiée.

Malgré ces défauts, ces approches présentent des fonctionnalités qui sont importantes pour la conception de systèmes robotiques et le recours aux OS temps-réels est certainement (systèmes de prototypage mis à part) la méthode de mise en œuvre la plus courante.

2) Formalismes et langages dédiés: Même si la mise en œuvre de la plupart des systèmes robotiques repose sur l'utilisation de ces OS temps-réels, il existe des formalismes dédiés et adaptés soit aux aspects discrets soit aux aspects continus.

a) Automates et SED: Le séquencement de tâches est classiquement réalisé en utilisant des méthodes reposant sur des automates d'états finis ou des réseaux de Petri. Si l'intégration de tels systèmes est rarement décrite [36], la mise en œuvre de ces systèmes ne repose pas toujours sur des techniques "classiques". De plus en plus, on constate l'apparition de techniques plus formelles reposant la plupart du temps sur le formalisme des systèmes dynamiques à événements discrets [30] (ou DEDS dont la formalisation est en grande partie due à Ramadge et Wonham [28]). L'utilisation en (vision) robotique d'un tel formalisme [2], [35], [19] permet la synthèse de comportements complexes des différents agents visuels impliqués. Kŏseká [19] propose d'utiliser les DEDS pour gérer le comportement d'un robot mobile. Des comportements élémentaires (navigation, évitement d'obstacle,...) sont modélisés par des automates très simples. Puis, ces automates sont ensuite composés, en utilisant le formalisme proposé par Ramadge et Wonham, synthétisant le comportement complexe du système. Notons que l'utilisation des DEDS autorise la vérification d'un grand nombre de propriétés dynamiques sur ces systèmes (vivacité, atteignabilité, etc...). De telles vérifications ont seulement été réalisées dans le cadre de systèmes robotique où seule la conception du superviseur est réalisée en utilisant ce formalisme (la conception des lois de commande n'étant pas prise en compte).

b) Functionnal reactive programming: Concernant là encore la gestion du comportement, Hager, Peterson et Hudak, partant du principe que les langages fonctionnels et en particulier le paradigme du "functionnal reactive programming"

[1]Nous ne traiterons pas ici du niveau planification. Notons juste qu'il est souvent intégré dans le niveau tâche.

(FRP) sont très bien adaptés à la mise en œuvre de la dualité continue/événementiel ont proposé dans un premier temps un meta langage FROB ("Functional ROBotics") reposant sur le langage Haskell [26]. Cette approche a été utilisée tant pour les aspects commande [26] que vision [29]. Pour améliorer la portabilité du concept de FRP, une extension permettant d'utiliser ce concept en C++ à par ailleurs été proposée dans [12].

c) Langages synchrones: Le concept de FRP se rapproche de l'abstraction proposée dans [21], [32] reposant sur l'utilisation d'un langage synchrone flot de donnée (Signal [20]) pour la mise en œuvre des processus continus (lois de commande) et sur la notion d'intervalle de temps pour la gestion des processus événementiels [31]. D'autres langages synchrones comme Esterel sont plus adaptés aux aspects discrets et sont utilisés dans des environnements spécialisés comme Orccad.

3) Environnements spécialisé: D'autres approches (dites hybrides) sont finalement apparues et se révèlent sans doute plus adaptées à l'intégration de tels systèmes. Elles ont été développées pour prendre en compte les défauts des méthodes précédentes. L'architecture hybride permet de considérer d'une part la conception des tâches de bas niveau (continues et flot de données) et de réaliser de manière efficace les contrôleurs de plus haut niveau (niveau mission). Parmi ces systèmes on retrouvera CONTROLLSHELL développé à Stanford [33], le système du LAAS [1] (et reposant, entre autre, sur Keops) et ORCCAD developpé à l'INRIA [34], [3]). D'autres systèmes ont par ailleurs été décrits dans [11] bien que, souvent, dans des contextes qui ne relèvent pas du processus de perception.

Orccad est un exemple significatif d'un tel système hybride [3]. Orccad est un système d'aide à la conception et à la programmation des systèmes robotiques dont les domaines d'application sont très vastes (citons par exemple la commande de robot mobile par asservissement visuel [27], [10] ou la commande de robots sous-marins [34]). L'objet de base du système Orccad est la tâche robot (*Robot-Task*) dont la fonction est de spécifier et de mettre en œuvre des actions robotiques simples. Le niveau application (spécifiant des actions complexes) est alors obtenu en composant les tâches-robots à l'aide de différents modes de synchronisation. L'objet final ainsi obtenu est appelé Procédure-Robot (*"Robot-Procedure"*). Chaque tâche robot est caractérisée par un aspect fonctionnel relatif aux lois de commande (*"Module-Task"*) et un aspect logique relatif au comportement du robot en réaction aux événements venant du monde extérieur pendant l'exécution des boucles de contrôle. L'exécution d'une loi de commande s'effectue si un ensemble de préconditions est vérifié et se termine si un ensemble de postconditions est vérifié. La spécification du système réalisée avec Orccad est ensuite automatiquement traduite dans le langage synchrone Esterel. Cette traduction permet d'obtenir un unique automate pour l'ensemble de l'application à partir duquel des propriétés sur le comportement du système peuvent être démontrées [17]. Notons que la modularité d'un système comme Orccad permet de prendre en compte relativement aisément l'hétérogénéité des plates-formes robotiques.

Cependant, seul le niveau application et la partie comportementale des tâches robot sont traduits dans ce langage. La partie loi de commande des tâches robot n'étant pas implémentée à l'aide d'un langage synchrone. Le source généré en Esterel est ensuite compilé vers des systèmes temps-réels (VxWork, QNX ou RTAI).

III. OUTILS DE PROTOTYPAGE RAPIDE

Si les environnements évoqués dans la partie précédente permettent, dans une certaine mesure, un développement sécurisé et fiable des systèmes robotiques, il n'en demeure pas moins qu'ils restent complexes à maîtriser et leur utilisation se limite le plus souvent à la mise en œuvre de systèmes industriels et non de prototypes (OS temps-réels mis à part).

Une seconde classe d'outils très largement utilisés dans notre communauté est donc celle des outils de prototypage rapide. La plupart des bibliothèques permettant un développement rapide de tâche robotique sont écrites en Matlab (éventuellement avec Simulink). Matlab est évidement un outils intéressant puisque très riche, très simple d'utilisation et permet au concepteur de se focaliser principalement sur les aspects méthodologiques et théoriques sans se préoccuper (dans un premier temps) des aspects d'implémentation. De plus l'utilisation de Simulink et ses "diagrammes de blocs" est très adaptée (comme les langages flot de données) à la mise en œuvre des lois de commande. En outre, on trouve désormais des bibliothèques Matlab permettant de gérer ou de simuler la cinématique des robots ou des processus de perception. Il reste que cet environnement est beaucoup plus adapté à la simulation qu'au développement sur des plates-formes expérimentales. D'autre bibliothèques écrites en C/C++ sont dans ce cas beaucoup plus adaptées. L'utilisation de langages objets et en particulier des mécanismes d'héritage permet de gérer de manière efficace l'hétérogénéité des matériels (robots, capteurs,...) tout en gardant un code générique. Précisons que l'utilisation de telles bibliothèques couplée avec celles de systèmes (OS) temps-réels reste possible. Nous nous retrouvons dans ce cas dans le contexte évoqué dans la section II-B.1. De même, ces outils peuvent servir à mettre en œuvre les lois de commande, charge à d'autres systèmes (Orccad par exemple) de prendre en compte, de manière sécurisée, les aspects missions [18].

Il existe un nombre très important de bibliothèques de ce type, dans le cadre de cet article, et à titre d'exemple, nous nous intéresserons plus particulièrement au cas particulier des environnements logiciels pour l'asservissement visuel.

A. Environnement logiciel pour l'asservissement visuel

Les techniques d'asservissement visuel [13] consistent à utiliser les informations fournies par un capteur de vision pour contrôler les mouvements d'un système dynamique, ce système pouvant être réel dans le cadre de la robotique, ou bien virtuel dans le cadre de l'animation d'entités artificielles ou de la réalité augmentée. Les applications robotiques sont et restent les applications privilégiées de l'asservissement visuel :

positionnement de robots, préhension, manipulation et suivi d'objets sont les principales tâches réalisables par asservissement visuel dans le domaine de la robotique manufacturière ou d'intervention (nucléaire, spatial, sous-marin, voire médical).

Cependant de nouvelles applications, très différentes, sont récemment apparues en dehors du domaine de la robotique : la réalité augmentée, la réalité virtuelle et l'animation avec la commande des déplacements d'une caméra virtuelle dans un environnement virtuel. Pour tous ces domaines, la mise à disposition au développeur d'une application de librairies logicielles permettant la construction modulaire et rapide de tâche d'asservissement visuel est importante.

Parmi les logiciels de ce type à vocation généraliste et pouvant servir de brique de base au développement de tels systèmes on retrouvera la Matlab robotics Toolbox [8] qui permet la création, la simulation et la manipulation de robots séries et Roboop [14] qui permet de réaliser des simulations élémentaires (grâce à une bibliothèque C++) de robots manipulateurs. Parmi les bibliothèques dédiées pouvant être utilisées pour l'asservissement visuel nous retiendrons principalement ViSP [22] et EGT [23]. Il n'y a pas cependant, à notre connaissance, de systèmes concurrents ayant toutes les caractéristiques de ViSP (voir section III-B). La simulation de lois de commande d'asservissement visuel peut bien sûr être réalisée en utilisant des outils disponibles sous Matlab (eg, Matlab robotics Toolbox [8], visual servoing toolbox for MATLAB / Simulink [4]), mais ce genre de simulation se prête mal à une implémentation sur des robots réels. Les bibliothèques de vision par ordinateur et de traitement d'images sont plus nombreuses (XVision [15], The Epipolar Geometry Toolbox [23], The Machine Vision Toolbox [9], Intel Vision ToolBox, ...) mais pas toujours adaptées aux problématiques de vision temps-réel.

B. ViSP "for visual servoing"

ViSP est une plate-forme logiciel développée à l'INRIA Rennes (IRISA) dans le projet Lagadic [22]. Les raisons ayant menés au développement de cette plate-forme logicielle sont multiples :

- mutualiser les développements réalisés en asservissement visuel au sein de la communauté de recherche ;
- faciliter les développements dans ce domaine, aussi bien pour poursuivre des activités de recherche, que pour en permettre une meilleure diffusion dans le monde industriel ;
- fournir à la communauté un outil commun d'expérimentations et de validation.

1) Mise en œuvre: La bibliothèque doit donc permettre des développements multi plates-formes (indépendants du matériel tant au niveau de l'informatique [type d'ordinateur, OS, caméras, etc.] que des robots utilisés), rapides et fiables de tâches d'asservissement visuel. Elle doit aussi permettre de réaliser les mêmes tâches d'asservissement visuel à la fois en simulation et sur manipulateurs réels (y compris en prenant en compte les contraintes temps-réels) en utilisant le même (ou presque le même) code. Les langages orientés objets permettent de prendre en compte ces contraintes de manière assez naturelle et C++ a donc été choisi pour implémenter ViSP. Par exemple, il est ainsi très simple en dérivant la classe `robot` générique de créer une interface avec tout nouveau robot (voir par exemple ceux de la figure 1).

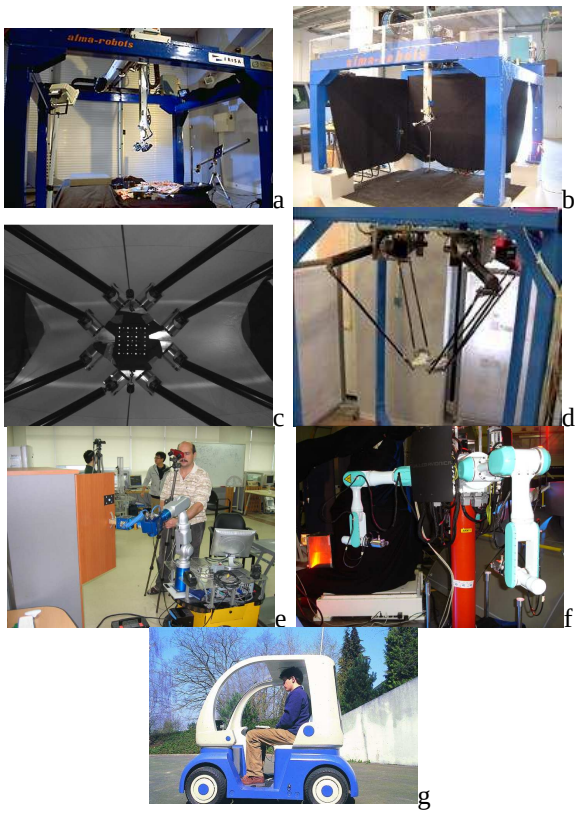

Fig. 1. Quelques robots ayant utilisé ViSP. (a-b) Robot portique cartésien de l'Irisa et du Lasmea, (c d) robots parallèles du Lirmm, (e) robot bras sur base mobile de Sungkyunkwan University (Corée), (f) Prototype de Eurobot (Estec, ESA), (g) véhicule Cycab de l'Irisa.

2) Architecture centralisée de ViSP: L'architecture logiciel sous-jacente est une architecture de type centralisé où la fonction principale est la tâche d'asservissement visuel (i.e, une boucle fermée sur les données images). En ce sens ViSP gère plutôt l'aspect continu du processus de vision robotique tel que nous l'avons défini plus tôt. L'utilisation d'un tel logiciel n'est cependant pas incompatible avec l'utilisation d'environnements logiciels de plus haut niveau et plus sécurisés. Ainsi dans le cadre du projet Vimanco pour l'agence spatiale européenne (ESA), l'utilisation de ViSP a été couplée à celle d'ORCCAD pour la réalisation de la partie gérant les enchaînements des tâches d'asservissement visuel [18].

Pour assurer la portabilité et pour pouvoir constamment étendre les fonctionnalités du logiciel, la plate-forme a été divisée en plusieurs modules : un module assurant la modélisation de l'information visuelle (primitives visuelles), un module permettant le calcul des lois de commande et gérant les contrôleurs des robots, un module regroupant les traitements d'images, les algorithmes de vision par ordinateur et de suivi. Enfin des modules annexes mais indispensables per-

mettent la visualisation, le calcul numérique, etc. La figure 2 résume l'architecture générale du logiciel et les dépendances entre modules.

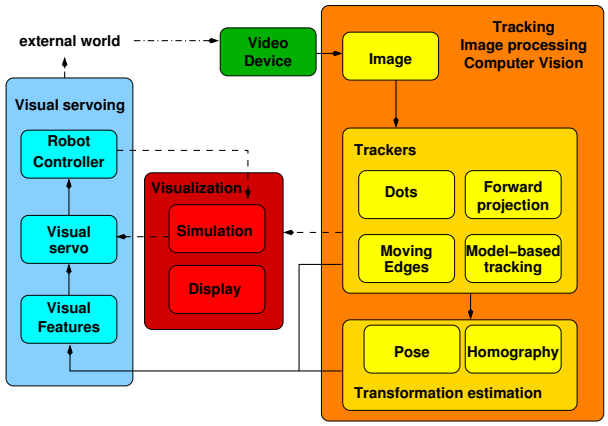

Fig. 2. Architecture logicielle de ViSP .

3) Utilisation de ViSP: Actuellement, la plate-forme ViSP fonctionne sous Linux, OSX, Solaris et Windows XP et a été utilisée avec succès sur un dizaine de robots différents (voir figure 1). Les tâches envisagées sont des tâches de positionnement, préhension, poursuite de cible, navigation, etc (voir Figure 3).

C. EGT

Comme pour ViSP, l'objectif de la librairie EGT (Epipolar Geometry Toolbox) développée à l'université de Sienne [23] est le prototypage rapide d'algorithmes d'asservissement visuel. La librairie se focalise principalement sur les aspects de modélisation géométrique du processus de vision et permet en particulier de considérer des systèmes monoculaires ou stéréoscopiques mais aussi les systèmes catadioptriques.

Contrairement à ViSP dont l'objectif affiché est de s'interfacer avec des robots réels, EGT est utilisé exclusivement en simulation. La librairie est un ensemble de fonctions Matlab et repose sur la Matlab robotics Toolbox [8] pour la gestion de la cinématique des robots. Les lois de commande peuvent être facilement mises en œuvre via Simulink.

D. Visual servoing toolbox

La visual servoing toolbox développée par E. Cervera [7] a des objectifs similaires à EGT mais est plus focalisée sur les aspects loi de commande. Elle repose elle aussi sur l'utilisation de Matlab et de Simulink. Elle a été écrite à des fins pédagogiques (cours Euron) et ne semble malheureusement pas avoir évolué depuis 2003. Un environnement de simulation JaVISS [6] en est dérivé. Écrit en Java, il permet, là encore à des fins pédagogiques, de simuler des tâches d'asservissement visuel élémentaires. Le système est très fermé et ne permet pas de modifier les lois de commande ou les primitives visuelles.

Une extension de la visual servoing toolbox a été décrite dans [5] mais ne semble pas être disponible pour les autres laboratoires.

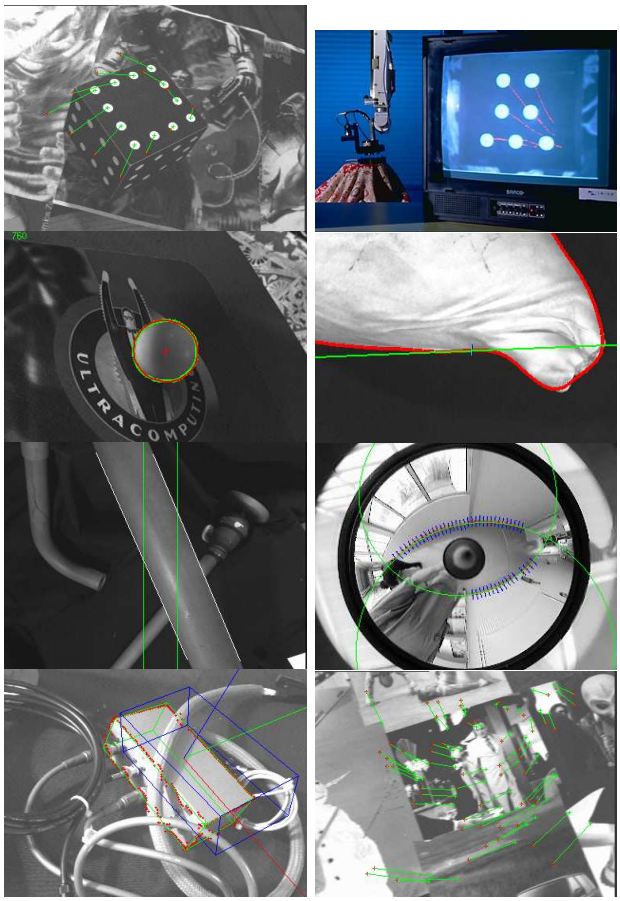

Fig. 3. Différentes tâches d'asservissement visuel mises en œuvre avec ViSP.

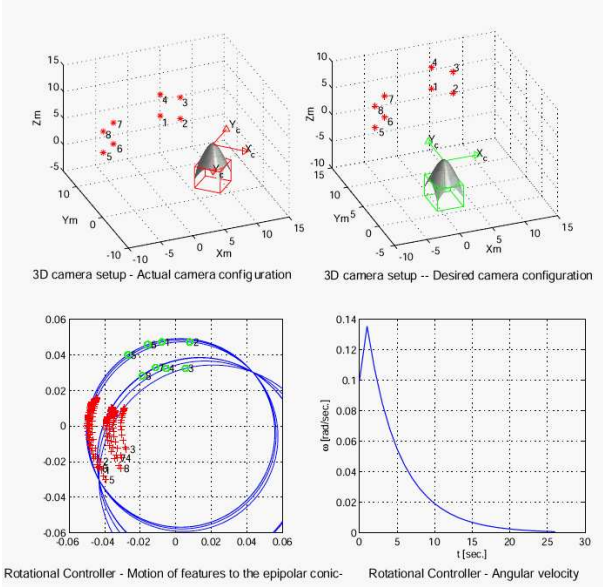

Fig. 4. Utilisation d'EGT pour une expérience d'asservissement visuel avec une caméra catadioptrique (extrait de [23]).

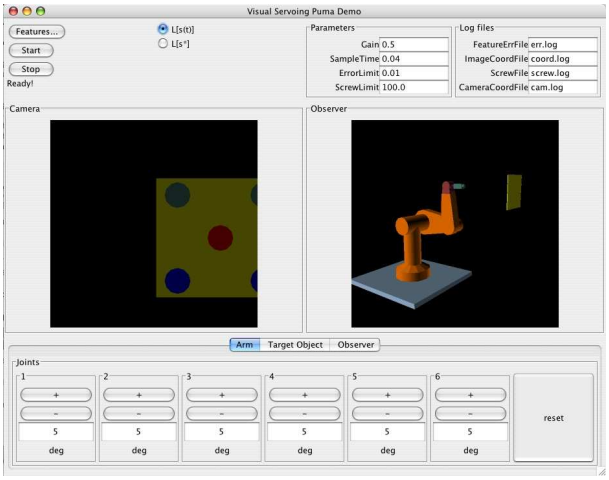

Fig. 5. JaVISS : simulation de tâche d'asservissement visuel à des fins pédagogiques [6].

E. Conclusion

Précisons que l'ensemble de ces logiciels Matlab robotics Toolbox [8], Visual servoing plateform (ViSP [22]), Epipolar Geometry Toolbox (EGT [23]) et la Visual servoing toolbox [4] sont des logiciels libres (soumis aux licences GPL ou QPL) et sont gratuitement utilisables et modifiables pour les utilisateurs. Bien qu'en théorie il soit possible de coupler les librairies Matlab à un robot réel il semble que seul ViSP soit utilisé à la fois pour le prototypage, la simulation et l'exécution sur des plates-formes robotiques réelles.

Précisons finalement qu'une étude a été menée à Yale University sur un système appelé ServoMatic [37] qui, couplé à XVision [15], aurait pu être comparable à ViSP mais ce système n'a jamais été réellement développé (un système similaire a cependant été proposé dans [12] mais n'est pas distribué).

IV. Conclusion

Malgré l'existence d'environnements adaptés à une mise en œuvre sécurisée de systèmes robotiques tant au niveau tâche que mission (ou même planification) leur utilisation reste pour le moment peu aisée et le recours à des outils qui devraient être dédiés au prototypage rapide est encore relativement courant. D'un coté nous avons des outils permettant de gérer de façon efficace le parallélisme implicite de ces systèmes et la synchronisation entre les différents processus. Ces mêmes outils permettent aussi, dans une certaine mesure, de vérifier que l'implémentation est conforme aux spécifications. De l'autre coté nous avons des bibliothèques qui présentent l'avantage principal d'être faciles à utiliser, le plus souvent multi plates-formes, s'adaptant à des systèmes hétérogènes et souvent libres (GPL).

References

[1] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An architecture for autonomy. *The Int. Journal of Robotics Research*, 17(4) :315–337, April 1998.

[2] Y. Aloimonos, E. Rivlin, and L. Huang. Designing visual systems : Purposive navigation. In Y. Aloimonos, editor, *Active Perception*, pages 47–102. Lawrence Erlbaum Assoc. publishers, Hillsdale, New Jersey, 1993.

[3] J.-J. Borrelly, E. Coste-Manière, B. Espiau, K. Kapellos, R. Pissard-Gibollet, D. Simon, and N. Turro. The orccad architecture. *The Int. Journal of Robotics Research*, 17(4) :338–359, April 1998.

[4] E. Cervera. Visual servoing toolbox for matlab / simulink. `http ://vstoolbox.sourceforge.net/`, 2003.

[5] E. Cervera. Distributed visual servoing : A cross-platform agent-based implementation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'05*, pages 3676–3681, Edmonton, Canada, August 2005.

[6] E. Cervera. Visual servoing toolbox for matlab / simulink. `http ://www.robot.uji.es/research/projects/javiss`, 2005.

[7] E. Cervera, A. P. del Pobil, F. Berry, and P. Martinet. Improving image-based visual servoing with three-dimensional features. *Int. Journal of Robotics Research*, 22(10-11) :821–840, 2003.

[8] P. Corke. A robotics toolbox for matlab. *IEEE Robotics & Automation Magazine*, 3(1) :24–32, September 1996.

[9] P. Corke. The machine vision toolbox. *IEEE Robotics and Automation Magazine*, 12(4), dec 2005.

[10] E. Coste-Manière, B. Espiau, and D. Simon. Reactive objects in a task level open controller. In *IEEE Int. Conf. on Robotics and Automation*, volume 3, pages 2732–2737, Nice, France, May 1992.

[11] E. Coste-Manière and B. Espiau (Eds). Special issue on integrated architecture for robot control and programming. *Int. Journal of Robotics Research*, 17(4), April 1998.

[12] X. Dai, G. Hager, and J. Peterson. Specifying behavior in c++. In *IEEE Int. Conf. on Robotics and Automation, ICRA'02*, volume 1, pages 153–160, Washington DC, USA, May 2002.

[13] B. Espiau, F. Chaumette, and P. Rives. A new approach to visual servoing in robotics. *IEEE Trans. on Robotics and Automation*, 8(3) :313–326, June 1992.

[14] R. Gourdeau. Object-oriented programming for robotic manipulator simulation. *IEEE Robotics & Automation Magazine*, 4(3) :21–29, September 1997.

[15] G. Hager and K. Toyama. The XVision system : A general-purpose substrate for portable real-time vision applications. *Computer Vision and Image Understanding*, 69(1) :23–37, January 1998. Also Research Report Yale University.

[16] D. Harel. *On the development of Reactive Systems*. Springer Verlag, New-York, USA, 1989.

[17] K. Kapellos. *Environnement de programmation des applications robotiques réactives*. PhD thesis, École des Mines de Paris, November 1995.

[18] K. Kapellos, F. Chaumette, M. Vergauwen, A. Rusconi, and L. Joudrier. Vision manipulation of non-cooperative objects. In *9th ESA Workshop on Advanced Space Technologies for Robotics and Automation, ASTRA 2006*, pages 279–286, Noordwijk, The Netherland, November 2006.

[19] J. Košeká, H. Christensen, and R. Bajcsy. Discrete event modeling of visually guided behaviors. *Int. Journal of Computer Vision*, 14(2) :179–191, March 1995.

[20] P. Le Guernic, M. Le Borgne, T. Gautier, and C. Le Maire. Programming real time application with signal. *Proceedings of the IEEE*, 79(9) :1321–1336, September 1991.

[21] E. Marchand, E. Rutten, H. Marchand, and F. Chaumette. Specifying and verifying active vision-based robotic systems with the signal environment. *Int. Journal of Robotics Research*, 17(4) :418–432, April 1998.

[22] E. Marchand, F. Spindler, and F. Chaumette. ViSP for visual servoing : a generic software platform with a wide class of robot control skills. *IEEE Robotics and Automation Magazine*, 12(4) :40–52, December 2005. Special Issue on "Software Packages for Vision-Based Control of Motion", P. Oh, D. Burschka (Eds.).

[23] G.L. Mariottini and D. Prattichizzo. Egt for multiple view geometry and visual servoing. *IEEE Robotics and Automation Magazine*, 12(4) :26–39, December 2005.

[24] L. Nana. Architectures logicielles pour la robotique. In *Journées nationales de la recherche en robotique*, pages 239–248, Guidel, October 2005.

[25] I.A. Nesnas. The clarary project : Coping with hardware and software heterogeneity. In D. Brugali, editor, *Software Engineering for Experimental Robotics*, volume 30 of *Springer Tracts on Advanced Robotics*, pages 31–70. Springer Verlag, 2006.

[26] J. Peterson, G.D. Hager, and P. Hudak. A language for declarative robotic programming. In *IEEE International Conference on Robotics and Automation, ICRA'98*, volume 2, pages 1144–1151, Leuven, Belgium, April 1998.

[27] R. Pissard-Gibollet. *Conception et commande par asservissement visuel d'un robot mobile.* PhD thesis, École des Mines de Paris, December 1993.

[28] P.J. Ramadge and W.M. Wonham. The control of discrete events systems. *Proceedings of the IEEE*, 77(1) :81–97, January 1989.

[29] A. Reid, J. Peterson, Hager. G., and Hudak. P. Prototyping real-time vision systems : an experiment in dsl design. In *international conference on Software engineering, ICSE'99*, pages 484–493, Los Angeles, California, 1999.

[30] E. Rutten. A framework for using discrete control synthesis in safe robotic programming and teleoperation. In *IEEE Int. Conf. on Robotics and Automation, ICRA'2001*, pages 4104–4109, Seoul, Korea, May 2001.

[31] E. Rutten and P. Le Guernic. Sequencing of data flow tasks in signal. In *ACM SIGPLAN Workshop on Language, Compiler and Tool Support for Real-Time Systems*, Orlando, Florida, June 1994.

[32] E. Rutten, E. Marchand, and F. Chaumette. An experiment with reactive data-flow tasking in active robot vision. *Software - Practices & Experience.*, 27(5) :599–621, May 1997.

[33] S., Chen V., G. Pardo-Castellote, and H. Wang. Controlshell : A software architecture for complex electromechanical systems. *The Int. Journal of Robotics Research*, 17(4) :360–380, April 1998.

[34] D. Simon, B. Espiau, E. Castillo, and K. Kapellos. Computer-aided design of a generic robot controller handling reactivity and real-time controller issues. *IEEE Trans. on Control Systems Technology*, 1(4) :213–229, December 1993.

[35] M.T. Sobh and R. Bajcsy. Visual observation under uncertainty as a discrete event processs. In *IAPR Int. Conf. on Pattern Recognition, ICPR'91*, pages 429–432, The Hague, The Netherlands, August 1992.

[36] A.D.H. Thomas, M.G. Rodd, J.D. Holt, and C.J. Neill. Real time industrial visual inspection : A review. *Real Time Imaging*, 1(2) :139–158, June 1995.

[37] K. Toyama, G. Hager, and J. Wang. Servomatic : A modular system for robust positioning using stereo visual servoing. In *Int. Conf. on Robotics and Automation*, pages 2636–2643, Minneapolis, April 1996.