Développement logiciel pour le contrôle commande multiaxe réactif

Pierre Laguillaumie

UNIVERSITE DE POITIERS

ÉCOLE DOCTORALE MIMME

T H È S E

pour obtenir le grade de

Docteur de l'Université de Poitiers

Faculté des Sciences Fondamentales et Appliquées

Diplôme National - Arrêté du 25 mai 2016

Présentée par

Pierre LAGUILLAUMIE

Développement logiciel pour le contrôle commande multiaxe réactif

Du contrôle bas niveau au jumeau numérique connecté

Thèse dirigée par Jean-Pierre Gazeau et Vincent Padois,

soutenue le le 7 septembre 2023

Jury :

Président :	Emmanuel Grolleau	-	ISAE-ENSMA
Rapporteurs :	Stéphane Caro	-	CNRS (LS2N)
	Charles Lesire-Cabaniols	-	ONERA
Directeurs :	Jean-Pierre Gazeau	-	CNRS (Pprime)
	Vincent Padois	-	Inria
Examinatrice :	Aurélie Clodic	-	CNRS (LAAS)

Table des matières

Introduction

Ce travail s'inscrit dans une démarche globale de conception de systèmes mécatroniques. Cette activité est centrale dans les travaux de l'équipe RoBioSS de l'Institut Pprime depuis une vingtaine d'années.

Entre 2000 et 2010, l'équipe RoBioSS a développé des démonstrateurs de recherche pour la robotique humanoïde, pour la manipulation ou qui présentaient des architectures mécaniques innovantes. La figure 1 présente des exemples de systèmes réalisés à cette époque. Sur le plan de l'architecture de commande, chaque système était équipé d'une solution technologique différente pour le calculateur comme pour les étages de puissance. Pour les exemples de la figure,

— le petit système locomoteur Tidom était équipé d'un automate programmable industriel (PLC) et de cartes électroniques embarquées Maxxon,
— la main LMS avait un contrôleur de mouvement standalone (sans automate complémentaire) permettant de piloter seize axes moteurs,
— la commande du robot d'impression sur grandes dimensions était réalisée avec du matériel National Instruments.

Cette situation rendait difficile le maintien en condition opérationnelle et l'amélioration des systèmes ainsi que le partage des algorithmes qui étaient développés dans plusieurs langages et exécutés sur des cibles différentes.

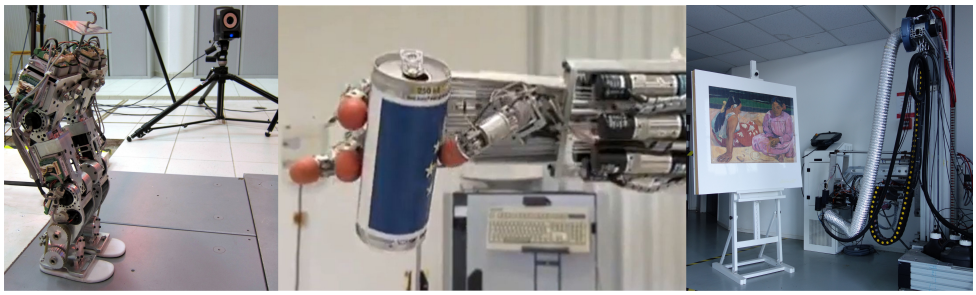


FIGURE 1 – Systèmes robotiques conçus avec des parties commandes hétérogènes.

Dans la décennie suivante, l'équipe a pris la décision de concevoir les parties commandes des nouveaux dispositifs avec des composants industriels standards et a entamé la création d'un environnement de développement logiciel à partir duquel tous les logiciels pourraient être réalisés. Cette évolution s'est accompagnée d'une diversification des centres d'intérêts scientifiques abordés par l'équipe. La figure 2 présente les exemples les plus caractéristiques de cette deuxième génération de systèmes : ORHRO, robot humanoïde de grande taille, le Delthaptic, une interface à retour haptique à grand espace de travail, un banc de caractérisation des arcs de compétition (un projet CREPS et FFTA), la main RoBioSS à seize degrés de liberté, le robot COMAU Racer3 du projet Harry2, lauréat de l'appel à projets COVR et Shiva, un dispositif d'apprentissage du geste humain.

Tous ces dispositifs sont basés sur la bibliothèque de contrôle multiaxe de l'équipe, mais chaque dispositif a un logiciel de commande différent.

Le travail présenté dans ce mémoire porte sur l'extension de ce travail d'unification du développement logiciel à l'aspect contrôle de robot. Le contexte de cette évolution est une accélération

FIGURE 2 – Systèmes conçus avec des composants de contrôle-commande standards.

du nombre et de la diversité des systèmes robotiques et des bancs de test à réaliser, avec des contraintes de temps de développement exigeantes. Le développement d'un nouveau système doit être compatible avec la durée d'une thèse de doctorat. La figure 3 présente les dispositifs fonctionnels ou en cours de développement dont les logiciels ont été développés dans le cadre de mes travaux de thèse.

Ainsi la question scientifique traitée ici porte sur la conception d'une architecture logicielle permettant de contrôler des systèmes robotiques ou multiaxes en interaction avec un environnement dynamique. Cette propriété est partagée par tous les dispositifs réalisés dans l'équipe, qu'ils soient des bancs d'essais, des ergomètres, des mains robotiques pour la manipulation fine ou la préhension avec évaluation des efforts d'interaction ou des systèmes téléopérés ou évoluant en environnement variable ou incertain.

Les contraintes de conception que nous avons retenues sont les suivantes :

— Le système de contrôle-commande doit être réactif, surveiller en permanence l'environnement du robot et réagir aux changements détectés dans un délai qui permet de réaliser ses tâches ou de les interrompre de façon sûre. Notre domaine d'application se limite à celui de systèmes déterministes présentant un fonctionnement temps réel à l'échelle de temps de la milliseconde.

— Une deuxième contrainte porte sur le choix de minimiser les développements électroniques spécifiques et de construire les armoires de commande à partir de composants d'automatisme général.

— Une troisième exigence porte sur la possibilité de transférer les résultats obtenus vers des applications industrielles. Cette exigence se traduit par l'adoption des standards industriels actuels de l'Industrie 4.0 ; elle a également des conséquences sur la décomposition fonctionnelle du logiciel pour que le développement puisse être partagé avec des programmeurs de différents niveaux de compétence.

Cet objectif et ces contraintes nous ont conduit à proposer une démarche de conception logicielle différente des principales méthodes existant dans la littérature et des outils offerts par

les solutions industrielles. Cette approche structurée part du constat que, pour construire un système réactif et déterministe, il est nécessaire de bien connaître le fonctionnement et les performances atteignables par les calculateurs et les cartes de puissance. Pour exploiter au mieux ces composants, nous avons choisi d'intégrer une partie importante des fonctions de pilotage des robots dans les automates programmables ou les PC industriels qui sont toujours présents dans la chaîne de commande des machines de production.

Le premier chapitre de ce manuscrit présente un état de l'art des intergiciels de robotique et présente le cahier des charges retenu pour la réalisation de notre environnement logiciel de contrôle multiaxe. Le deuxième chapitre introduit la bibliothèque *rtrmac* et une méthode de développement d'objets logiciels pour les systèmes de contrôle-commande industriels qui pilotent des axes de mouvement. La partie suivante décrit la démarche mise en place pour étendre cette bibliothèque à la coordination multiaxe orientée robotique. Enfin, le quatrième chapitre présente des exemples du développement logiciel de dispositifs complexes qui peuvent présenter plusieurs calculateurs et sont associés à des jumeaux numériques.

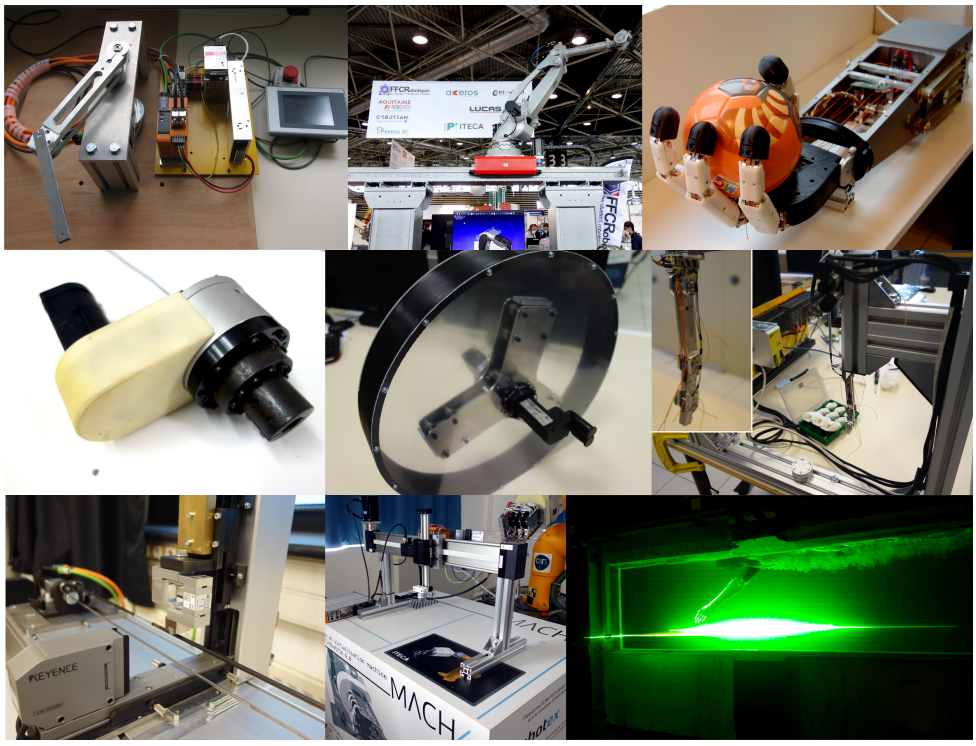


FIGURE 3 – Systèmes développés avec une architecture logicielle unifiée.

Liste des principales publications scientifiques

Revues internationales avec comité de rédaction

— *Lower Limb Sensor-to-Segment Calibration for Joint Kinematics Analysis With Inertial Measurement Units : Is There an Ideal Method ?* L. Pacher, C. Chatellier, P. Laguillaumie, R. Vauzelle, L. Fradet. IEEE Sensors Journal, vol. 22, no. 22, pp. 22148-22160, 15 Nov.15, 2022, doi : 10.1109/JSEN.2022.3209883.

— *ORHRO : OPEN Robotics Humanoid RObot - from a mechanical retrofit to a new real time industrial controller based walking robot.* A. Eon, J. Gastebois, P. Laguillaumie, P. Vulliez, P. Seguin, J. P. Gazeau. Computer Methods in Biomechanics and Biomedical Engineering. (2019) 22 :sup1, S502-S504 DOI : 10.1080/10255842.2020.1714997

— *RTRobMultiAxisControl : A Framework for Real-Time Multiaxis and Multirobot Control.* H. Fischer, M. Vulliez, P. Laguillaumie, P. Vulliez, J. P. Gazeau. IEEE Transactions on Automation Science and Engineering. (2019) doi :10.1109/TASE.2018.2889813

— *Focus on the mechatronics design of a new dexterous robotic hand for inside hand manipulation.* P. Vulliez, J. Gazeau, P. Laguillaumie, H. Mnyusiwalla, P. Seguin. Robotica, Vol. 36(8), 1206-1224. (2018) doi :10.1017/S0263574718000346

Conférences internationales avec comité de sélection

— *Learning an arm movement on a mechatronic device : influence of visual feedbacks occurrence.* R. Crolan, A. Eon, Pierre Laguillaumie, A. Decatoire, C. Scotto, Y. Blandin. ACAPS (2021)

— *Proprioceptive contribution to learning an elbow flexion-extension task.* R. Tisserand, M. Vaurs, A. Decatoire, A. Eon, P. Laguillaumie, C. Scotto, Y. Blandin. ACAPS (2021)

— *Comparison of calibration methods for accelerometers used in human motion analysis.* A. Nez, L. Fradet, P. Laguillaumie, T. Monnet, P. Lacouture. Proc. of the 22nd Congress of the European Society of Biomechanics (2016)

Liste des brevets et dépôts logiciels

Brevets déposés

— *Brevet n°2010813 du 22/10/2020 pour : « Dispositif de rééducation, d'entraînement ou de préparation physique »*, CNRS, Université de Poitiers. Inventeurs : A. Decatoire, A. Eon, P. Laguillaumie

— Brevet n°1914606 du 17/12/2019 pour "Préhenseur marin sensible en efforts", CNRS. Inventeurs JP. Gazeau, P. Vulliez, P. Laguillaumie, C. Mizera

— *Brevet n°1909724 du 04/09/2019 pour « Main robotique »*, CNRS. Inventeurs : JP. Gazeau, P. Vulliez, P. Laguillaumie, C. Mizera

Dépôt du logiciel : *« RTRobMultiAxisControl »* à l'Agence de Protection des Programmes, 19 juillet 2018. IDDN. FR.001.300012.000.S.P.2018.000.31235.

Chapitre 1

Approches et outils du contrôle robotique

La robotique industrielle est apparue au début des années 1960 avec la mise en œuvre du premier robot industriel à 6 axes chez General Motors [Nof 1999]. Le robot « Unimate » a été installé en 1961 dans l'usine de Trenton, dans le New Jersey. L'application consistait pour le robot à sortir des pièces fraîchement embouties de la presse à chaud et à les plonger dans un bain de refroidissement. Sur ce modèle, la robotique industrielle s'est ensuite largement développée jusqu'au début des années 2000, dans tous les domaines de l'industrie. Le robot industriel a ainsi remplacé massivement l'être humain pour la plupart des travaux pénibles et répétitifs. Du point de vue de la sécurité, ces robots industriels spécialisés pour une tâche donnée opèrent dans une cellule fermée hors de portée de l'humain en environnement connu, les seuls risques existants étant les défaillances techniques ou les erreurs humaines [Järvinen 1995], [Nagamachi 1986].

Au début des années 2000, cette association « Homme-robot » a été complétée avec l'apparition d'une nouvelle robotique non industrielle : la robotique de service et, en général, la robotique d'assistance humaine [Haidegger 2013]. La différenciation entre les robots industriels et les robots de service est basée sur leur domaine d'application. Grâce aux progrès de la technologie des capteurs et contrairement aux robots industriels, les robots de service peuvent opérer dans l'environnement de travail des humains et peuvent interagir avec eux. La norme ISO 13482 définit ainsi la robotique de service comme suit : « Robot qui effectue des tâches utiles pour les humains ou les équipements à l'exclusion d'une application d'automatisation industrielle ». Cette définition particulièrement large réunit des applications aussi diverses que l'assistance à domicile avec le robot aspirateur Roomba ou la tondeuse à gazon, le domaine militaire avec la plateforme de télémanipulation mobile PackBot, la santé avec le robot de chirurgie laparoscopique Da Vinci et le robot semi-humanoïde Pepper de SoftBank Robotics.

Parallèlement au développement de la robotique de service, une autre forme de robotique est en cours de développement. Il s'agit de la robotique collaborative, qui a pour objectif de permettre au robot d'interagir avec les travailleurs en production ou simplement de partager le même espace de travail. Les robots collaboratifs se distinguent de la robotique de service par leur domaine d'utilisation. La proximité des humains et des robots implique une probabilité plus élevée d'apparition de situations pouvant entraîner des blessures humaines [Caputo 2013] et l'intégration de ces nouvelles technologies sur les lignes de production est toujours un challenge scientifique et technique actuel.

Les travaux relatifs à ce mémoire s'inscrivent dans cette dynamique de coopération et de contrôle des composants de l'environnement de production. Ils portent sur les robots et d'une manière générale les machines, les systèmes polyarticulés actionnés avec un nombre d'axes élevé.

La première section de ce chapitre présente les évolutions récentes de la robotique industrielle pour prendre en compte des possibilités d'interaction avec des opérateurs humains. La partie

suivante distingue les différentes architectures de contrôle robotique et les composants matériels associés. Les principaux environnements logiciels conçus pour permettre une conception générique des logiciels de pilotage de robots sont ensuite passés en revue. Finalement, je présente le cahier retenu pour la réalisation d'un nouvel environnement logiciel intégré pour le pilotage de machines multiaxes.

1.1 De la robotique industrielle à la robotique collaborative

L'objectif de la robotique collaborative est d'associer les capacités de l'homme et du robot au sein d'un espace de travail partagé ou collaboratif. Cette union permet de combiner les qualités de chacun pour assister les opérateurs au poste de travail. Les atouts du robot sont les suivants : il peut produire des efforts importants et accompagner la manipulation de charges lourdes ; sa précision est de l'ordre de 0,05 mm et il est capable de travailler sans s'arrêter. Quant à l'humain, sa dextérité, sa flexibilité et sa capacité à prendre des décisions lui permettent de s'adapter à une grande variété de tâches et d'événements imprévus.

[Briquet-Kerestedjian 2019] distingue trois types d'interaction Humain - robot, en fonction de la distance entre eux et du type d'informations échangées :

- *L'interaction Homme-robot à distance*, téléopération ou télémanipulation, désigne les applications dans lesquelles un humain opère le robot à distance (dans le cas de la téléopération nucléaire ou chirurgicale) ou avec un décalage temporel dû à la durée d'échange des informations entre les deux agents (par exemple, dans le cas de robots d'exploration).
- *L'interaction cognitive Homme-robot* traite des facteurs sociaux et cognitifs qui définissent les échanges entre les robots et les humains. Ces échanges sont réalisés de façon verbale (commandes vocales, reconnaissance vocale, traitement du langage naturel) ou non verbale (reconnaissance de gestes, suivi du regard de la tête et des yeux, activité des bio-signaux).
- *L'interaction physique Homme-robot* est caractérisée par la proximité spatiale entre le robot et l'opérateur qui partagent un espace de travail commun, appelé espace de collaboration. Dans ce cas, l'opérateur a la possibilité d'entrer en contact physique avec le robot en fonctionnement, pour réaliser des tâches ensemble ou que le robot fournisse une assistance physique complémentaire.

Ainsi, et à titre d'exemple, la robotique collaborative offre un compromis entre l'assemblage manuel, et l'assemblage entièrement robotisé. Ces avantages et inconvénients sont mis en lumière dans [Fryman 2012].

Le développement de cette nouvelle technologie est régi par les normes internationales de sécurité ISO 10218-1 et ISO 10218-2 qui spécifient les exigences de sécurité relatives à la conception des robots industriels et des systèmes robotiques, respectivement. Elles présentent quatre types d'opérations collaboratives (figure 1.1), à savoir : l'arrêt nominal de sécurité (mode 1), le guidage manuel (mode 2), la surveillance de la vitesse et de la séparation humain-robot (mode 3), la limitation de la puissance et de la force (mode 4) [Villani 2018].

Les trois premiers modes de collaboration ne nécessitent pas nécessairement l'utilisation d'un robot collaboratif. Dans ce cas, la sécurité de l'homme repose sur des dispositifs de sécurité qui garantissent que le robot est arrêté à proximité de l'homme (modes 1 et 3) ou que le mouvement du robot est lié à une entrée de commande directe de l'homme (mode 2). Ces modes utilisent des composants de sécurité et exigent de mettre en œuvre une méthodologie de développement

du logiciel de sécurité qui doivent garantir qu'il n'y a pas contact direct inattendu entre le robot et l'homme.

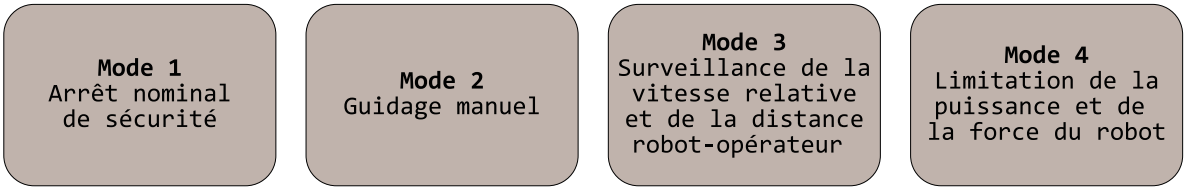


FIGURE 1.1 – Les quatre modes de collaboration humain-robot

Le terme *cobot* est associé au mode de collaboration limité en puissance et en force (mode 4). Avec ce mode, le contact entre le robot et l'opérateur est possible, que ce soit intentionnellement ou non. Ce type de collaboration nécessite que le robot soit capable de percevoir les forces d'interaction avec l'environnement. Le contact doit être parfaitement compris et contrôlé afin de minimiser les risques.

La figure 1.2 présente des robots collaboratifs de référence, dotés de capacités de détection des forces, de 6 ou 7 degrés de liberté et d'une charge utile pouvant atteindre 25 kg. La plupart des robots collaboratifs sont dotés de fonctions intégrées qui garantissent les exigences de sécurité ci-dessus [El Zaatari 2019].

FIGURE 1.2 – Principaux cobots commercialisés : Franka Production 3 ©Franka Emika, LBR iiwa ©Kuka, CRX-10 ©Fanuc, UR16 ©Universal Robot, APAS ©Bosch, AURA ©Comau.

A titre d'exemple, les robots LBR iiwa et Franka Emika sont équipés de capteurs de couple en sortie de réducteur sur les sept axes. L'évaluation des couples articulaires des robots d'Universal Robot est basée sur la détection du courant de l'actionneur et l'ajout d'un capteur de force à l'effecteur final permet au robot d'interagir en toute sécurité avec l'homme. Certains robots industriels sont équipés de capteurs complémentaires pour étendre leurs domaines d'usage : le robot COMAU AURA est équipé de capteurs tactiles pour détecter les contacts [Hoshi 2006], le robot BOSCH APAS possède une peau capacitive qui détecte la proximité d'un humain et s'arrête avant le contact.

Deux enjeux essentiels se dessinent aujourd'hui dans la mise en œuvre d'applications robotiques en environnement ouvert exploitant ces modes de collaborations avec des robots industriels

et/ou collaboratifs : la sécurisation de l'interaction entre le robot et son environnement et la robustesse de l'application.

1.1.1 Robustesse de l'application robotique

L'émergence de la robotique collaborative ne signifie pas la fin de la robotique industrielle. Les deux types de robotique vont coexister et travailleront ensemble afin de résoudre les problèmes de flexibilité et de personnalisation du flux de production.

Un besoin de flexibilité accrue pour de nouvelles applications industrielles impose de développer des applications robotiques avec un contrôle multirobot en temps réel dans un environnement ouvert et dynamique. Potentiellement ces applications peuvent impliquer des robots industriels, des robots collaboratifs et des humains dans une zone de travail partagée pour lesquelles un haut niveau de coordination et d'interopérabilité peut être requis.

Un tel défi ne peut être relevé que si tous les composants mécatroniques du processus de production sont connectés et capables de communiquer de manière sûre. La chaîne des composants logiciels doit également être à même d'implémenter des mécanismes de communication et de coordination pour garantir le déterminisme du comportement de l'application robotique, sa sûreté de fonctionnement tout en sécurisant les interactions entre les robots et les humains dans un environnement dynamique.

Dans ce contexte, on peut définir la robustesse d'un système de contrôle-commande par l'association de plusieurs propriétés :

— il doit participer à garantir la stabilité du système qu'il contrôle,
— il doit être tolérant vis à vis des variations et des perturbations de l'environnement,
— il doit être réactif aux erreurs : il doit être capable de détecter et de gérer les erreurs qui surviennent, qu'elles soient dues à des défaillances de la partie opérative ou à des usages non prévus du logiciel. Il devrait être doté de mécanismes de détection d'erreurs, de techniques de diagnostic et de stratégies de récupération pour minimiser les conséquences des erreurs qui se produisent en production.

Dans le contexte du contrôle-commande, on peut définir la stabilité d'un système comme sa capacité à assurer ses fonctions sans discontinuité et sans danger sur toute sa durée de vie, quelques soient les sollicitations extérieures rencontrées.

Il s'agit d'être en mesure de pouvoir supprimer les défaillances à même de mettre l'application robotique en défaut. La suppression des défaillances implique d'être en mesure de sécuriser le fonctionnement de l'application robotique.

Les robots sont susceptibles de connaître des défaillances dues à des problèmes de matériel, de logiciel ou de communication. Ainsi dans la littérature sur les réseaux complexes et le contrôle multirobot, de nombreuses approches ont été proposées pour aborder le maintien de la communication avec des groupes de robots mobiles effectuant des tâches données. Les défaillances successives, en particulier pour les robots jouant un rôle central dans la topologie du réseau, peuvent conduire à un service inopérant ou réduit, ce qui a conduit au développement de méthodes originales pour assurer la continuité de service [Jakel 2011], [Ji 2007]. Des approches de contrôle ou de conception résilientes sont également développées pour permettre la récupération des états de défaillance. Dans ce contexte, on peut citer des travaux concernant les robots volants [Briod 2014], souples [Tolley 2014], auto-configurables [Yim 2007] et mobiles [Zhang 2017]. Dans [Portugal 2021], afin de contribuer à la résilience des robots, les auteurs mettent l'accent

sur la nécessité de proposer une architecture modulaire, dans laquelle tous les composants ont des interfaces standard pour interagir entre eux et rendre le système globalement résilient. La résilience se distingue de la robustesse par la définition proposée dans [Zhang 2010], la résilience y est définie par « la capacité du système à poursuivre son fonctionnement au niveau souhaité lorsqu'il subit un dommage partiel ».

Sur la base de ces définitions, nous cherchons, à travers nos développements, à contribuer à rendre les applications robotiques robustes par rapport aux contraintes fortes liées à la dynamique de l'environnement. En particulier, nous travaillons à garantir la stabilité de leurs performances pour des dispositifs pouvant impliquer un nombre élevé de systèmes polyarticulés actifs (axes pilotés, machines, robots).

1.1.2 Sécurisation de l'interaction

On peut classifier les stratégies de sécurisation de l'interaction robot-environnement en deux catégories comme illustré par la figure 1.3. Ces stratégies sont envisageables pour éviter d'attenter à l'intégrité physique des personnes, ou a minima, de réduire les risques de blessures ou leur gravité. On distingue ainsi [Ikuta 2003] :

— les stratégies de pré-collision qui visent en temps réel soit à anticiper la collision, soit à réduire la force de l'impact ,

— les stratégies de post-collision qui visent à réduire les conséquences de l'impact.

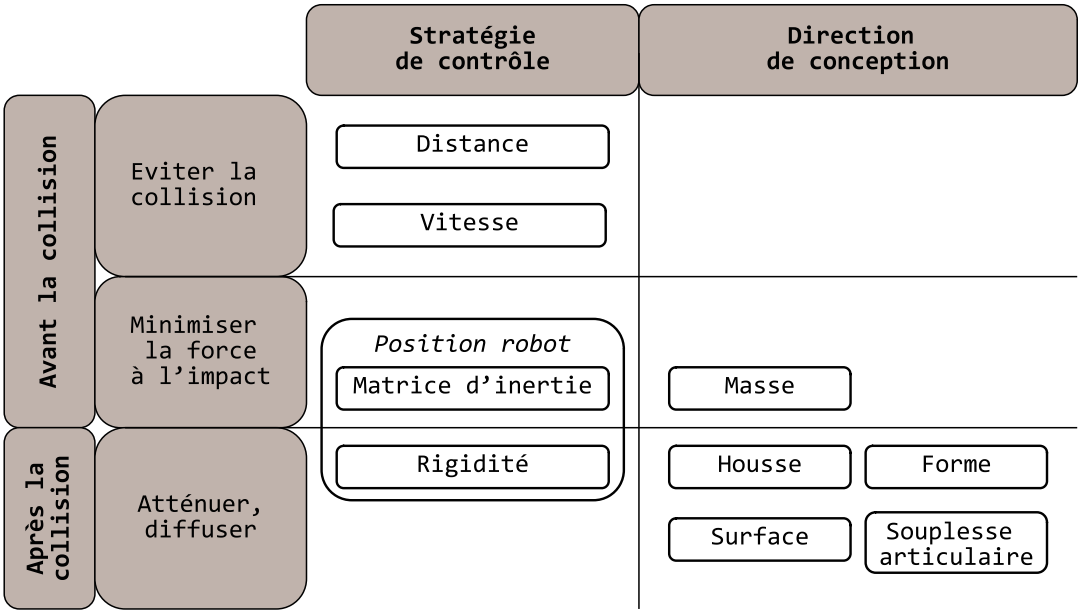


FIGURE 1.3 – Classification des stratégies de prévention des risques dans la collaboration homme-robot , selon [Ikuta 2003].

On observe ainsi que la stratégie de contrôle complète le travail sur le design de la partie

opérative, dans la mise en œuvre d'une stratégie de pré-collision. Dans ce cas, l'objectif est de minimiser la quantité de mouvement et la raideur des éléments qui vont se percuter. La stratégie de contrôle contribue dans une moindre mesure dans la mise en œuvre d'une stratégie de post-collision : le temps de réaction du système de contrôle ne permettra pas au robot de modifier son comportement durant les instants suivant l'impact, mais peut diminuer les effets de l'écrasement des membres du sujet humain.

La stratégie de conception est donc prépondérante dans la mise en œuvre d'une stratégie de post-collision. Il s'agit dans ce contexte de proposer des approches de conception [De Santis 2008] permettant de diminuer les conséquences de l'impact telles que :

— l'allègement de la structure du robot et un design avec une enveloppe externe du robot adaptée évitant les arêtes vives [Park 2011], l'évaluation de l'ergonomie du cobot et son application au design de ce dernier [Maurice 2017] ;
— le développement de peaux passives [Ulmen 2010], [Pang 2021], [Cheng 2019] ;
— le développement d'actionneurs à raideur variable [Tonietti 2005] et plus généralement à impédance variable [Vanderborght 2013].

La stratégie de contrôle est déterminante pour prévenir toute situation de risque associée au mouvement du robot dans son environnement. De nombreuses stratégies ont ainsi été implémentées pour sécuriser l'interaction robot-environnement en amont. La perception de l'environnement est essentielle dans ce contexte. Classiquement le robot mobile est équipé de capteurs de proximité (infrarouges, lasers, capacitifs, à ultrasons) qui sont exploités pour produire une planification réactive du mouvement en fonction de l'évolution dynamique de l'environnement [Laconte 2019]. La perception partielle de l'environnement propre à cette technologie limite souvent leur utilisation à proximité de l'humain avec l'utilisation de grilles d'occupation 2D de l'environnement [Hoffmann 2016]. D'autres travaux récents relatifs à la collaboration homme-robot proposent d'équiper les opérateurs de marqueurs passifs sur leur vêtement de travail en exploitant des caméras dédiées à la capture de mouvement pour capter finement l'ensemble des mouvements de l'opérateur [Laconte 2021]. De nombreux travaux récents s'appuient sur l'utilisation de capteurs permettant une perception 3D de l'environnement (LiDAR, caméras Time of Flight, caméras RGB-D) [Otani 2018], [Fuseiller 2019].

Pour réaliser ces stratégies de contrôle de sécurisation de l'interaction du robot, il est essentiel de pouvoir les implémenter et les déployer dans un environnement logiciel et une architecture de contrôle intégrés. Il faut à la fois assurer une continuité entre les différents niveaux de contrôle d'une cellule robotisée et sécuriser le processus de perception-action.

1.2 Architectures de contrôle en robotique

L'architecture de contrôle d'un système robotique est la clé de voûte de la qualité du comportement de l'application robotique et de la qualité de réalisation de la tâche visée. Depuis l'émergence du premier robot industriel en 1961, la littérature n'a eu de cesse de proposer des architectures de contrôle spécifiques à un domaine (robotique mobile, robotique aérienne, robotique collaborative), d'essayer de les classer, de proposer des solutions dites ouvertes, modulaires, généralisables à tout type d'application. Il apparaît encore aujourd'hui que la variété des propositions ne permet pas encore d'adresser l'un des enjeux fondamentaux d'une application robotique : garantir le processus élémentaire de perception-action.

Pour cela, il nous semble nécessaire de répondre aux exigences suivantes :

— prendre en compte la dynamique de l'environnement,
— sécuriser l'interaction du robot avec l'environnement,
— coordonner le contrôle de plusieurs robots ou machines au sein d'un même système automatisé,
— pouvoir implémenter des stratégies de contrôle de la littérature comme la commande robuste, par exemple.

Les enjeux propres au développement d'une architecture de contrôle robotique sont alors les suivants :

— interopérabilité entre les composants matériels d'une cellule robotique : robots, machines, dispositifs de perception,
— gestion temporelle déterministe du comportement de l'application robotique,
— ouverture de l'environnement de développement s'appuyant sur des langages de programmation non propriétaires,
— facilitation du transfert hors du cadre académique, industrialisation et passage en production.

La proposition d'architecture de contrôle que nous développons dans ce mémoire dépasse volontairement le simple processus de perception-action d'un seul robot. Une application robotique peut en effet comporter une variété de robots agissant de manière coordonnée afin de converger vers un objectif conjoint au sein d'une même cellule.

Dans la suite de cette section, nous distinguons les différents types d'architectures de contrôle de machines avant d'illustrer les solutions proposées par plusieurs fabricants de robots.

1.2.1 Classification des architectures de contrôle

Le paradigme « sense-plan-act » [Siciliano 2008], [Murphy 2000] illustré par la figure 1.4 décrit le rôle dévolu à l'architecture de contrôle. Il s'agit ainsi de :

— percevoir l'état du système et de l'environnement en exploitant les capteurs proprioceptifs et extéroceptifs mis en œuvre au sein de la cellule robotique,
— traiter les informations de perception afin de déterminer les actions à produire,
— produire les actions déterminées par l'envoi des consignes résultantes aux actionneurs.

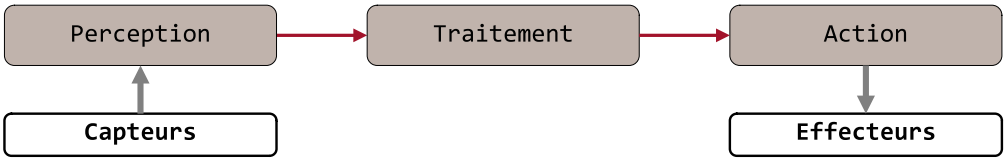


FIGURE 1.4 – Le paradigme « sense-plan-act ».

Pour répondre aux enjeux de ce paradigme, la littérature s'est attachée à proposer des architectures et à les classifier ; l'objectif étant finalement d'essayer d'effectuer un découpage fonctionnel du processus de perception-traitement-action. On distingue ainsi plusieurs types de classifications de ces architectures avec des qualificatifs différents qui doivent être définis, afin de les distinguer. On considère généralement trois types d'organisation de ces architectures

[Russell 2010], [Passama 2014] :

— les architectures hiérarchiques ou délibératives,
— les architecture réactives ou comportementales,
— les architecture hybrides.

1.2.1.1 Les architectures hiérarchiques ou délibératives

Par architecture hiérarchique, on entend généralement une structuration de l'architecture en programmes ou modules distincts assurant une fonctionnalité. Ces programmes communiquent entre eux d'une manière prévisible et prédéterminée. Cette hiérarchisation se traduit par des contraintes temporelles différentes entre les modules et classiquement par une décomposition en trois couches :

— une couche décisionnelle qui planifie l'enchaînement des actions relatives à la tâche robotique spécifiée, cette couche est également qualifiée de couche délibérative [41] dans la mesure où elle doit proposer des solutions pour produire des tâches parfois complexes en termes de planification ;
— une couche exécutive qui supervise l'exécution des actions ;
— une couche fonctionnelle ou bas niveau, en liaison directe avec la couche physique (capteurs et actionneurs).

Cette structuration en trois couches permet d'identifier des temporalités différentes entre les couches qui peuvent aller de quelques minutes sur la couche décisionnelle, à quelques millisecondes de temps de cycle pour la couche fonctionnelle.

1.2.1.2 Les architectures réactives ou comportementales

L'architecture réactive est directement inspirée du vivant. Le principe est de pouvoir définir sur la base d'informations capteurs spécifiques des comportements spécifiques. A noter que ces comportements peuvent être facilement décrits dans des machines d'état dédiées. On peut citer les travaux de Brooks [Brooks 1986] qui font référence pour ces architectures. Dans cette architecture de contrôle, des comportements ou règles sont définis à différents niveaux de contrôle. Pour exemple le premier niveau met en œuvre l'évitement d'obstacles ; le deuxième niveau met en œuvre un déplacement vers un point particulier ... À chaque cycle d'exécution, des règles sont activées en fonction de l'état actuel du robot et des entrées des capteurs. Parmi les règles activées, le niveau de contrôle d'une règle détermine sa priorité : la règle de plus haut niveau est déclenchée. Dans la plupart des cycles, une règle de bas niveau est activée, alors que, plus rarement, une règle de niveau supérieur est activée.

1.2.1.3 Les architectures hybrides

L'architecture hybride associe architecture hiérarchique et architecture réactive et vise à accroître le temps de réaction du robot à l'aide de comportements prédéfinis. On peut citer l'exemple de l'architecture du robot autonome AuRA [Arkin 1997], qui résume l'esprit propre à ces architectures. L'architecture du robot AuRA traite le problème de la planification comme deux systèmes distincts qui s'interfacent l'un avec l'autre : un planificateur global et un planificateur local réactif. On peut faire l'analogie avec l'homme ; en effet des preuves psychologiques et neurophysiologiques de la coexistence de deux systèmes de planification distincts chez l'homme

sont rapportées dans [Norman 1986], ce qui a inspiré les roboticiens dans la mise en œuvre de telles architectures.

Parmi ces architectures de contrôle, nombreuses sont celles proposées dans le contexte de la navigation de robots mobiles dans des environnements non structurés avec des contraintes fortes de coordination et de prise en compte de la dynamique de l'environnement.

1.2.2 Vers une architecture idéale

D'autres approches de classification sont proposées dans la littérature. Ainsi dans [Mouad 2014], les auteurs classifient les architectures de contrôle selon un autre point de vue. Ils choisissent en effet d'opposer :

— d'une part, les architectures de contrôle centralisées versus les architectures distribuées,
— d'autre part, les architectures de contrôle cognitives versus les architectures réactives.

Cette proposition de classification et la très grande variété des publications relatives à des architectures de contrôle robotique spécifiques à chaque application illustre clairement la nécessité de réaliser une décomposition fonctionnelle très fine des différentes fonctionnalités, tâches et sous-tâches requises dans le développement d'une application robotique.

Les questions de la hiérarchisation, de la concurrence entre les tâches, de l'implémentation de modes de contrôle réactifs ou cognitifs, de la temporalité des tâches et du déterminisme temporel associé, de la distribution des tâches sur plusieurs cibles ou de la centralisation du contrôle doivent pouvoir être adressées dans un cadre architectural plus large d'un point de vue logiciel offrant davantage de flexibilité et d'ouverture que les architectures proposées pour contrôler un ensemble de robots.

Sur la base des architectures existantes, l'architecture de contrôle « idéale » d'une application robotique doit ainsi permettre à la fois :

— de hiérarchiser les tâches et de leur affecter des niveaux de priorité,
— de définir des temporalités différentes entre les tâches,
— de pré-définir des tâches spécifiques,
— de distribuer les tâches sur plusieurs cibles (PC, automates programmables ou microcontrôleurs) ;
— d'offrir une communication temps réel entre tous les composants matériels et logiciels (tâches, sous-tâches) de l'application.

1.2.3 Architectures de contrôle industrielles

La production d'une architecture logicielle de contrôle pour la robotique ne peut être envisagée sans prendre en compte les contraintes propres au transfert hors du cadre purement académique. Pour ce faire, nous rappelons les caractéristiques associées aux contrôleurs des robots industriels et à leur mise en œuvre. La norme ISO 8373 :2012 2 fournit une spécification du vocabulaire lié aux robots et composants robotiques, en faisant référence à l'industrie. Un robot y est défini comme un « manipulateur multi-application reprogrammable commandé automatiquement, programmable sur trois axes ou plus, qui peut être fixé sur place ou mobile, destiné à être utilisé dans des applications d'automatisation industrielle ».

Le robot est constitué d'une partie « opérationnelle » ou « opérative » qui réalise la tâche et d'une partie « décisionnelle » ou « commande » qui contrôle la partie opérative.

La partie opérationnelle est constituée de la structure mécanique du robot, des actionneurs, de l'instrumentation proprioceptive (codeurs intégrés, mesures de couple) et extéroceptive (caméras, télémètres, efforts d'interaction), de l'effecteur (torche de soudage, préhenseur, pistolet de peinture, ...).

La partie décisionnelle ou commande assure la gestion du pilotage des axes et leur régulation à partir des consignes transmises depuis l'interface de programmation. La figure 1.5 présente un contrôleur industriel standard. Il s'agit d'une baie Fanuc R30iB qui contient un calculateur principal, une carte de sécurité, des cartes de puissance pour l'alimentation des moteurs et un périphérique d'interaction avec l'utilisateur (teach pendant).

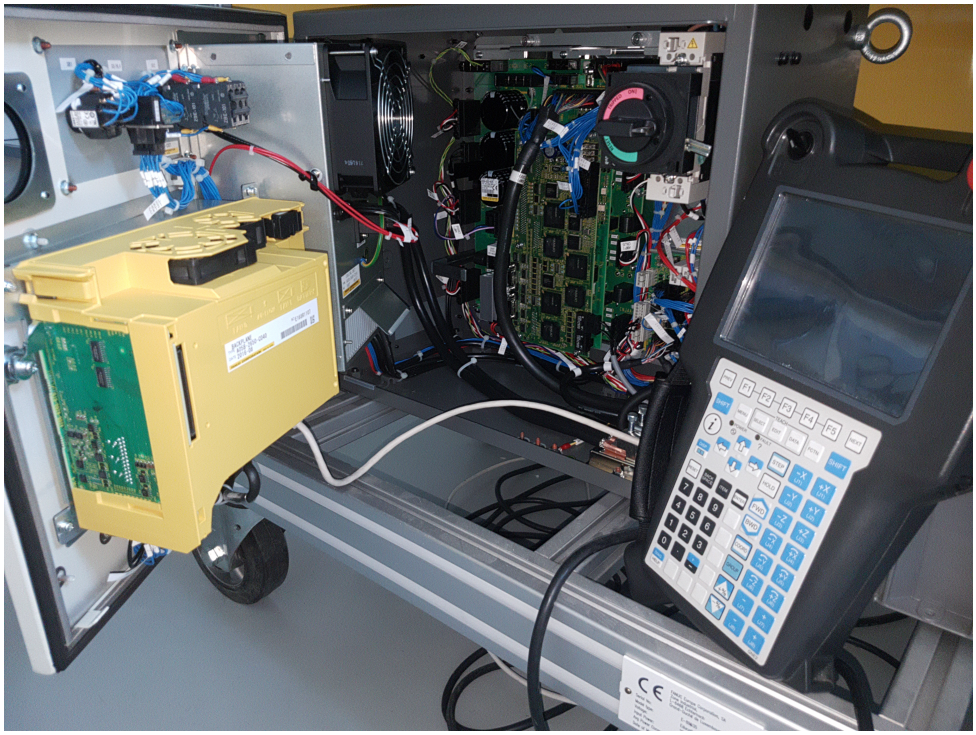


FIGURE 1.5 – Contrôleur de robot industriel 6 axes.

La baie R1C de Comau (cf. Figure 1.6) permet de commander la gamme des petits robots sériels quatre et six axes. Elle intègre, dans un encombrement très réduit un PC industriel, un automate de sécurité, deux contrôleurs de mouvement à trois axes chacun, une carte de communication pour protocole Ethernet industriel et une carte électronique intégrant des fonctions de sécurité (cf. Figure 1.6). Contrairement à l'armoire de commande Fanuc, celle-ci est constituée de composants industriels standards (à l'exception de la carte de sécurité).

Il est possible de constituer une partie commande de robots à partir de composants standards. Cette approche permet une mise à l'échelle simplifiée d'une cellule robotique puisque le calculateur principal peut être mutualisé pour piloter plusieurs robots des axes de mouvement complémentaires. A titre d'exemple, la figure 1.7 montre les architectures matérielles des armoires de commande de deux dispositifs utilisés par l'équipe RoBioSS. La première contrôle un robot industriel 6 axes, la seconde un groupe de deux robots 6 axes entraîné par un axe linéaire. Certains constructeurs utilisent le terme de « robotique ouverte » pour désigner cette solution

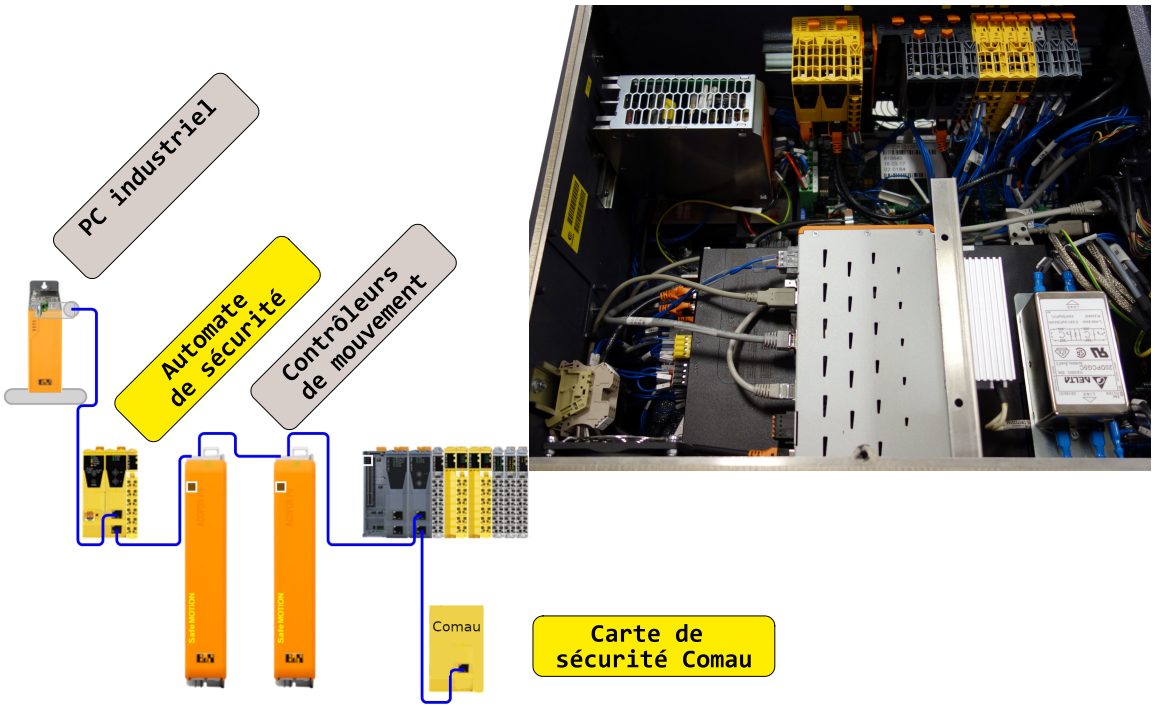


FIGURE 1.6 – Composants de la baie Comau R1C.

qui permet de piloter des architectures mécaniques différentes avec des armoires de commande composées de composants électroniques d'usage général.

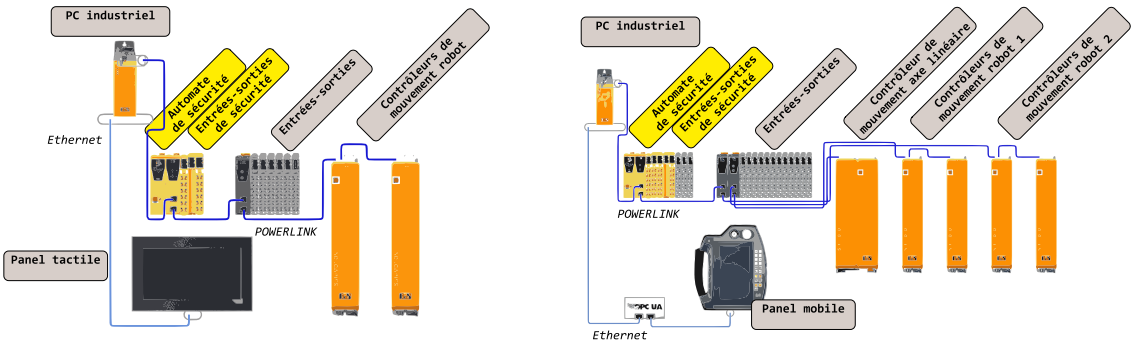


FIGURE 1.7 – Architectures matérielles de l'armoire de commande de la cellule robotique du projet HARRY2 (gauche) et de la cellule SMALA (droite).

Pour résumer, le contrôleur d'un robot industriel intègre généralement les composants suivants :

— les contrôleurs d'axes : les contrôleurs d'axes permettent d'asservir les actionneurs du robot avec les boucles de contrôle associées (régulation en cascade classique d'une boucle de régulation avec régulateur de courant, régulateur de vitesse et régulateur de position) s'appuyant sur la mesure de courant interne aux actionneurs et la mesure des codeurs

intégrés aux actionneurs. Généralement un contrôleur d'axes assure la régulation d'un, deux ou trois axes en fonction des solutions matérielles choisies ;
— un automate programmable industriel (PLC) : le PLC permet d'assurer le contrôle coordonné des axes en position et en vitesse aussi bien dans l'espace articulaire que dans l'espace opérationnel. Il intègre les machines d'état permettant de gérer les différents modes de marche et d'arrêt du robot ;
— la gestion de la sécurité : cette gestion est généralement assurée par une carte de sécurité ou un automate de sécurité associant des composants de sécurité (barrière matérielle ou immatérielle, arrêts d'urgence, interrupteurs de sécurité, relais de sécurité,...) de manière à prendre en compte toute demande d'arrêt ou tout fonctionnement imprévu. Les caractéristiques des éléments de sécurité varient selon le niveau d'interactivité nécessaire entre l'homme et le robot. Ces différents éléments de sécurité sont mentionnés dans la norme NF EN ISO 10218-2. En fonction de l'application robotique et de l'analyse des risques associée, différents éléments de sécurité peuvent être actifs selon le mode de fonctionnement du robot ;
— un ordinateur de type PC avec environnement Windows ou Linux permettant de gérer l'interface homme-machine avec l'utilisateur et intégrant l'environnement de programmation (IDE) propre au constructeur ;
— un boîtier d'apprentissage, panneau de contrôle ou « Teach Panel » qui permet à l'opérateur de piloter directement le robot en mode manuel dans l'espace articulaire ou opérationnel, d'effectuer un certain nombre d'opérations de configuration (apprentissage des repères outil, des repères de travail, calibration, ...) ou d'arrêter le robot.

Au niveau logiciel, l'architecture de contrôle repose sur :
— les contrôleurs d'axes configurés logiciellement en fonction des contraintes applicatives : les mouvements des axes sont ainsi pilotés à travers un bus de communication temps réel depuis le PLC. L'architecture logicielle du PLC offre un fonctionnement temps réel dur avec un déterminisme temporel strict basé généralement sur un système d'exploitation temps réel tel que VxWorks, ou plus récemment des versions de Linux pour le temps-réel. L'implémentation des tâches cycliques périodiques de bas niveau au sein du PLC permet :
  — de gérer les différents modes de déplacements des axes en implémentant généralement les fonctionnalités propres au standard PLCopen Motion Control,
  — les différents modes de déplacement du robot dans l'espace articulaire, dans l'espace opérationnel avec l'implémentation des modèles cinématiques direct et inverse,
  — l'interprétation des lignes de commande issues du programme dans le langage robot du fabricant.
— Le programme propre à l'application est quant à lui mis en œuvre au sein généralement d'un PC intégré au contrôleur. Le développeur dispose d'une suite logicielle permettant le développement des programmes applicatifs, leur débogage et leur simulation. La réalisation des programmes utilisateurs est réalisée avec un langage de programmation propriétaire (Fanuc KAREL, Stäubli Val3, KUKA KRL, ABB RAPID) ou avec un langage standard pour certains cas spécifiques (Java pour le Kuka IIWA). Dans ce contexte, l'absence de standard pour la programmation des robots industriels est un facteur limitant dans le développement d'applications multirobots multifabricants. Plusieurs stratégies sont en train d'émerger pour lever cette difficulté. Ce point sera développé dans le chapitre 3.1.1.

Un point important dans le choix d'un robot est la possibilité de pouvoir accéder au contrôleur du robot depuis une ligne de communication externe afin de l'intégrer dans une machine ou une ligne de production. Les modes d'accès au contrôleur du robot varient d'un fabricant à l'autre ; en fonction du bus de communication (temps réel ou non) proposé, du type d'accès offert (régulateur en position par exemple), du temps de cycle propre au protocole implémenté, les performances obtenues peuvent être extrêmement variables. Le tableau 1.1 illustre quelques exemples de caractéristiques relatives à l'accès à des contrôleurs de robots industriels et collaboratifs.

Universal Robot A titre d'exemple, les robots collaboratifs du fabricant Universal Robot offrent un accès à leur contrôleur depuis une communication TCP/IP et via les scripts de commande `URScript` avec un temps de cycle de l'ordre de 100ms. En l'occurrence, même si le temps de cycle du contrôleur de position interne des robots UR est de 8ms, l'accès à l'intergiciel de pilotage *ROS* implémenté sur un ordinateur distant devient problématique car il existe un délai de 170ms entre la réception de la consigne et la position actuelle retournée [Ravn 2014], [Andersen 2015]. Plusieurs travaux ont été proposés dans la littérature exploitant ce mode de communication sous ROS pour de la téléopération [Wieczorek 2020], [Omarali 2017]. Une analyse de l'erreur entre la position souhaitée et la position réelle est fréquemment utilisée pour évaluer la qualité du contrôle téléopéré ; l'erreur étant liée à la vitesse, des mouvements plus rapides entraînent une erreur de position plus importante. Aussi une évaluation du délai entre la pose commandée et la pose réelle offre un meilleur indicateur de la qualité du contrôle [Park 2016]. Ainsi dans [Wieczorek 2020], les auteurs observent un délai de 350ms entre pose commandée et pose réelle dans la réalisation d'une ligne droite avec un robot UR5. Dans [Omarali 2017], les auteurs relèvent l'impossibilité de recopier le mouvement humain acquis par un système de capture du mouvement en raison du délai entre pose commandée et pose réelle à travers les drivers ROS ou Matlab [de Gier 2015] mis en œuvre ; ce constat a conduit les auteurs à développer une commande prédictive avec un horizon de prédiction de 500ms. On note également la latence de 50ms évoquée dans la documentation technique de Shadow Robotics [Company 2021] dans l'envoi d'une commande au bras manipulateur UR10e exploitant un contrôleur CB3 et une communication Ethernet à travers l'intergiciel ROS. Pour autant, dans [Fritsche 2015], les auteurs estiment qu'un délai pouvant atteindre 800ms est acceptable dans le contexte de leur application. Ces résultats illustrent le fait que l'accès au contrôleur industriel des robots collaboratifs Universal Robot à des fins de commande externe offre une variété de possibilités, que ce soit via ROS et URScripts, via Matlab et URControl ou via C-API au niveau logiciel, via TCP/IP Real Time, via TCP/IP Socket, via Modbus TCP ou via Profinet/Ethernet IP au niveau de la communication [Ponsà Cobas 2020]. Toutefois à la lecture des travaux exploitant ces différents modes d'accès, les résultats et performances obtenus varient de manière significative et démontrent la difficulté d'uniformiser l'accès aux modes de contrôle du robot depuis un contrôleur externe avec du déterminisme temporel dans le comportement du robot.

Kuka On peut également citer les travaux relatifs au contrôle des robots industriels Kuka ; plusieurs interfaces logicielles ont été développées par différents laboratoires afin de s'interfacer avec les différentes familles de contrôleurs Kuka. La boîte à outils Kuka Control Toolbox (KCT) présentée dans [Chinello 2011], propose une collection de fonctions MATLAB permettant le contrôle du mouvement des robots industriels Kuka. A travers cette interface de programmation sous

Matlab, l'utilisateur peut piloter tous les robots Kuka sériels 6 axes de petite taille et de faible charge. KCT fonctionne sur un ordinateur distant connecté au contrôleur industriel KRC via TCP/IP. Un serveur multithread s'exécute sur le contrôleur et communique via Kuka.Ethernet KRL avec un client dont le rôle est de gérer l'échange d'informations avec le robot. On peut également citer le logiciel open-source OpenKC [Schopfer 2010] dédié au contrôle des fonctionnalités du robot Kuka LWR. OpenKC permet les mesures de force et de couple, ainsi que l'accès aux différents modes de fonctionnement à travers l'interface logicielle Kuka Robot Sensor Interface (RSI). L'interface FRI a également été développée pour les robots collaboratifs Kuka LWR [Schopfer 2010]. La communication basée sur FRI fournit un accès direct de bas niveau au contrôleur LBR4+ avec un temps de cycle pouvant atteindre 1ms et une mesure de la qualité de la communication s'appuyant sur un horodatage et le comptage des trames perdues. A travers cette interface et l'implémentation d'une communication basée sur une structure d'informations et de commande, il est possible d'exploiter depuis un PC/PLC distant les modes de contrôle en position articulaire ou cartésien ainsi que les modes de contrôle impédants cartésien et articulaire du robot Kuka LWR. On peut citer un dernier moyen de contrôler les robots Kuka en temps réel. Dans [Sanfilippo 2015], les auteurs ont développé un mode de contrôle des robots Kuka basé sur un serveur nommé KukaVarProxy développé avec le langage Microsoft Visual Basic 6.0 qui permet via l'interface Kuka CrossComm d'accéder directement aux variables bas niveau des contrôleurs industriels Kuka KRC. Nous avons, dans l'équipe RoBioSS, testé ces fonctionnalités, il s'agit d'un mode de communication qui exploite le fait que le client KukaCrossComm développé par Kuka pour établir la communication entre les programmes de l'IDE Kuka (langage KRL) sous Windows et le PLC de contrôle bas niveau offre la possibilité d'accéder aux variables systèmes du contrôleur bas niveau à travers cette interface ; il s'agit néanmoins d'une possibilité non documentée et non recommandée par le fabricant. Ainsi on observe également pour Kuka, la difficulté de proposer un accès uniformisé à ses différents contrôleurs et aux modes de contrôle associés.

Fanuc On peut effectuer les mêmes constats quant au contrôle d'accès distant aux contrôleurs industriels des robots Fanuc. L'accès distant est réalisé généralement à travers une communication Ethernet TCP/IP et au niveau logiciel côté contrôleur à travers un programme interprété en langage Karel. Le temps de cycle au sein du contrôleur peut descendre à 8ms (paramètre ITP InTerpolation Period configuré à travers la variable système $SCR.$ITP_TIME) selon le type de contrôleur Fanuc. Le moyen d'accès le plus commun est l'utilisation du driver fanuc_driver_exp via l'intergiciel ROS-Industrial. Les performances relevées dans la documentation du driver [van der Hoorn 2022] démontrent la difficulté à reproduire des trajectoires avec le contrôle précis des paramètres position-vitesse-accélération en continu. Le pilote est ainsi mieux adapté aux applications de type pick-and-place ; c'est-à-dire finalement la production de mouvements discrets dans lesquels on cherche à atteindre successivement des positions cibles. On peut également citer [Garg 2021] pour lesquels les auteurs ont mis en place un serveur de messagerie basé sur l'utilisation de sockets avec un programme en langage KAREL au sein du contrôleur Fanuc qui reçoit les paramètres articulaires depuis un client distant et partage les données d'état du robot avec son client tout en exécutant les mouvements consignés. Cette communication est mise en œuvre dans cet exemple afin de créer un jumeau numérique du robot Fanuc connecté à l'environnement de simulation Roboguide du constructeur.

Même si des correctifs et évolutions permettent d'améliorer, au gré des versions développées

par les chercheurs, ces performances, le manque de robustesse et de standardisation de l'accès aux contrôleurs industriels posent question. Ces quelques exemples illustrent clairement la complexité de l'uniformisation de l'accès aux contrôleurs industriels existants dans le contexte d'applications robotiques multirobots et multifabricants.

TABLE 1.1 – Exemples d'accès distants aux contrôleurs industriels de robots.

Fabricant / baie de commande	Communication / période [ms]	Temps réel ?	Type d'accès
Stäubli / Unival open controller	POWERLINK / 2	oui	Accès au régulateur de position
Universal Robot / CB2	TCP/IP / 100	non	Envoi - réception de scripts (URScripts)
Kuka / LBR4+	UDP - FRI, horodatage trames UDP / 1	non	Accès aux modes de contrôle articulaire, cartésien, impédant articulaire, impédant cartésien
Kuka / KRC4	TCP/IP - RSI / 10	non	Accès aux variables de contrôle internes au contrôleur

Dans le contexte de l'usine du futur où l'interopérabilité est un enjeu majeur, on conçoit que l'absence de standard de communication temps réel entre des robots de différents fabricants, que le fait d'avoir des modes d'accès différents aux boucles de contrôle des robots est de nature à limiter fortement le développement d'applications nécessitant un haut niveau de synchronisation entre les machines/les robots et une prise en compte de la dynamique de l'environnement pour des applications robotiques collaboratives impliquant à la fois des robots industriels (modes collaboratifs 1 à 3 de la norme ISO 10218, cf. § II) et des robots collaboratifs (modes collaboratifs 4 de la norme ISO 10218, cf. § II).

1.3 Intergiciels pour le contrôle en robotique

Comme nous l'avons évoqué dans la section précédente, la robotique industrielle avec une offre de robots, de contrôleurs et un environnement logiciel dédié répond aujourd'hui à la plupart des contraintes industrielles classiques propres à la production de tâches répétitives dans un environnement généralement fermé et parfaitement connu et invariant dans sa configuration.

En l'occurrence, dès lors qu'il faut intégrer des contraintes supplémentaires : interfacer des composants de nature différente avec les contrôleurs (capteurs de vision, d'efforts, interfaces haptiques, gants de données, préhenseurs évolués, contrôleurs externes, ...), développer une application robotique logicielle avec du déterminisme temporel dans l'ordonnancement, gérer l'exécution de l'ensemble des tâches de l'application répartie sur plusieurs cibles en adressant une grande variété de composants, il devient indispensable de pouvoir offrir une architecture logicielle capable de répondre à ces enjeux.

Le besoin de faire communiquer des applications s'exécutant sur plusieurs machines à travers

un réseau est apparu très tôt et a conduit au début des années 90, à l'apparition des premiers middlewares ou intergiciels. On pourra retenir la définition proposée dans [Bakken 2001] pour définir un middleware : « A layer of software above the operating system but below the application program that provides a common programming abstraction across a distributed system » ; c'est-à-dire une couche logicielle située entre système d'exploitation et programme d'application qui permet de proposer un cadre commun pour la programmation d'applications distribuées à travers le réseau sur plusieurs machines ou processeurs.

Dans ce contexte de nombreuses propositions d'intergiciels ont vu le jour dans la communauté robotique académique et industrielle afin de permettre le développement logiciel d'applications robotiques complexes et distribuées.

1.3.1 Classification des intergiciels

L'intergiciel est une solution aujourd'hui communément admise pour faciliter le développement d'applications distribuées hétérogènes. Le choix d'un intergiciel particulier devient ainsi une question de conception essentielle : comme il n'existe pas de solution de distribution « unique », ce choix n'est pas neutre et peut influencer profondément la conception finale et les performances d'une application. Avant de présenter les caractéristiques des intergiciels dédiés aux applications robotiques, il est important de rappeler les différentes classes d'intergiciels en fonction des mécanismes utilisés pour communiquer entre les composants :

— *Transactions Processing* : cette classe concerne l'exécution de transactions mettant en œuvre un mécanisme simple de coopération entre les clients, les serveurs et les systèmes de gestion des bases de données. Elle garantit le fait que la tâche est correctement récupérée en cas de défaillance et que la base de données reste cohérente même en cas d'arrêt et d'exécution incomplète. Des protocoles en deux ou trois phases sont mis en œuvre pour sécuriser ces transactions. Elle est clairement orientée vers les bases de données et ne rentre donc pas dans le cadre du sujet qui nous intéresse ;.

— *Remote Procedure Calls* (RPC) : cette classe de middleware distribue l'exécution de routines sur un réseau.

— *Object Request Broker* (ORB) : cette classe permet de distribuer et de partager les objets d'une application sur des réseaux hétérogènes ; elle constitue le cœur de l'architecture CORBA, assurant les communications et les synchronisations entre objets.

— *Message Oriented Middleware* (MOM) : cette classe permet aux applications distribuées de communiquer et d'échanger des données en envoyant et en recevant des messages.

— *Service Oriented Middleware* (SOM) / Entreprise Service Bus (ESB) : cette classe de middlewares permet de proposer une approche de développement orientée services basée sur l'architecture SOA (Service Oriented Architecture) avec l'objectif de rendre ces services accessibles et disponibles à travers des modèles et protocoles standards.

1.3.1.1 Middleware de type RPC

Une procédure RPC [Birrell 1983] représente une action déclenchée par un programme afin d'exécuter une sous-routine dans un autre processus, généralement sur un autre ordinateur partagé en réseau. Cette procédure est développée comme s'il s'agissait d'un appel de sous-routine local, sans la complexité de l'interaction à distance. Cela peut être vu comme une

architecture client-serveur où le client est le processus qui appelle l'exécution du sous-programme distant.

Ce type de middleware permet ainsi aux procédures d'une application d'appeler des procédures d'applications distantes comme s'il s'agissait d'appels locaux. Un middleware RPC offre ainsi la possibilité de mettre en œuvre facilement des procédures RPC par la génération du code logiciel « Stub » côté client et du code logiciel équivalent côté serveur « Skeleton », et il permet également la transformation de la représentation en mémoire d'un objet en un format de données adapté au stockage ou à la transmission entre différentes parties d'un programme informatique ou d'un ordinateur à un autre (« Marshalling / Unmarshalling » des données). Ce code logiciel exécutable « Stub » permet de gérer la procédure distante côté client et de récupérer localement le résultat de l'exécution distante du code logiciel exécutable « Skeleton » sur le serveur.

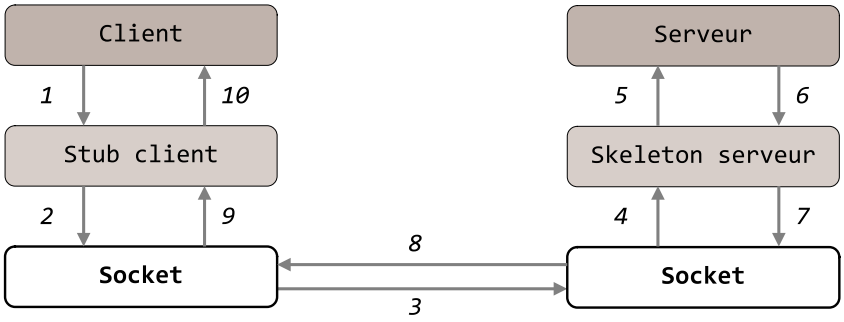


FIGURE 1.8 – Les différentes étapes d'un appel de procédure RPC.

Comme illustré sur la figure 1.8, l'appel d'une procédure RPC est initié par le client via le code logiciel exécutable « stub » qui envoie un message de demande à un serveur distant connu (étapes 1 à 5) pour exécuter une procédure spécifiée via le code exécutable logiciel « Skeleton ». Le serveur distant répond avec le résultat de l'exécution (étapes 6 à 10) et l'application poursuit son traitement. Il est important de noter que les procédures RPC sont synchrones ; ainsi le plus grand inconvénient est la possibilité que l'appel de procédure distante échoue à cause de problèmes de réseau imprévisibles [Qilin 2010].

L'intergiciel de type RPC met en œuvre un mécanisme de liaison qui localise les procédures distantes et les met à la disposition de l'appelant de manière transparente. Traditionnellement, ce type d'intergiciel traite les programmes basés sur des procédures ; il comprend désormais également des composants basés sur des objets.

1.3.1.2 Middleware de type ORB

Un middleware de type ORB (Object Request Broker) repose sur la notion de bus logiciel [Chacko 2013]. ORB permet ainsi la communication entre deux programmes se situant sur des machines différentes en ayant l'impression que l'appel reste local. Un courtier de message (Broker) est mis en œuvre pour gérer l'interaction entre le client et le serveur. L'architecture d'ORB la plus connue est CORBA (Common Object Request Broker Architecture). CORBA [Pope 1998] est né du consortium OMG (Object Management Group), créé en 1989 et composé d'environ 800

entreprises (Sun, Oracle, IBM, Hewlett Packard, IBM) avec l'objectif de faire émerger des standards pour l'intégration d'applications distribuées hétérogènes et la réutilisation de composants logiciels.

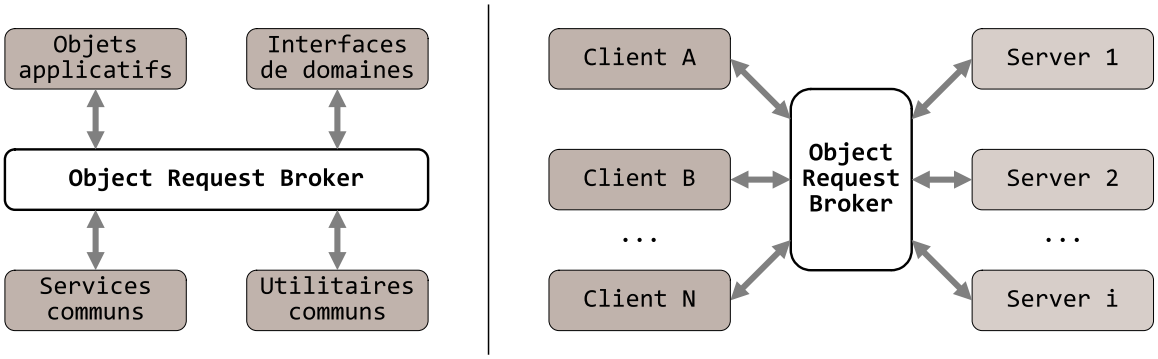


FIGURE 1.9 – Middleware de type ORB : architecture OMA (gauche), principe de l'ORB (droite).

Cette norme a été définie par l'OMG de 1990 (rédaction du guide relatif à l'architecture OMA (figure 1.9) : Object Management Architecture) à 1991 (spécification de CORBA V1.1 basée sur l'OMA). Au sein de l'architecture CORBA, tout est basé sur la notion d'objet. CORBA a été développé dans le but de permettre une interconnexion et une communication des logiciels entre eux, même s'ils sont implémentés sur des systèmes hétérogènes ou dans des langages différents. Le principe de fonctionnement client-serveur s'appuie sur le fait que le client demande au Broker l'exécution d'un service (demande d'exécution d'une méthode sur un objet). A l'issue de cette demande, le Broker identifie un serveur capable de fournir le service et il transmet la requête au serveur identifié (cf. figure 1.9).

1.3.1.3 Middleware de type MOM

Les MOM assurent une communication distribuée entre plusieurs machines et applications avec un mode d'échange client-serveur asynchrone entre émetteur et récepteur. Ce modèle non bloquant permet de résoudre un grand nombre de limitations propres aux middleware RPC. Les participants à un système basé sur MOM ne sont pas obligés de bloquer et d'attendre l'envoi d'un message. Après l'envoi d'un message, ils sont autorisés à poursuivre le traitement, ce qui permet la livraison de messages lorsque l'expéditeur ou le destinataire n'est pas actif ou disponible pour répondre au moment de l'exécution. L'envoi de messages dont la livraison peut prendre plusieurs minutes est ainsi possible, contrairement à des mécanismes tels que RPC.

Tous les MOM proposent ainsi un mécanisme de communication où les clients envoient des messages avec leurs demandes d'exécution de service à un ou plusieurs serveurs à travers le réseau avec en retour le résultat de l'exécution [Pope 1998]. On distingue toutefois deux classes de MOM avec des mécanismes différents pour le relais des messages :

— Les MOM basés sur les files d'attente : les MOM basés sur les files d'attente sont basés sur une architecture un-à-un comme le montre la figure 1.10. L'application réceptrice possède une file d'attente où elle stocke tous les messages reçus de toutes les applications clientes. Elle traite ensuite ces messages selon la méthode FIFO (First In First out).

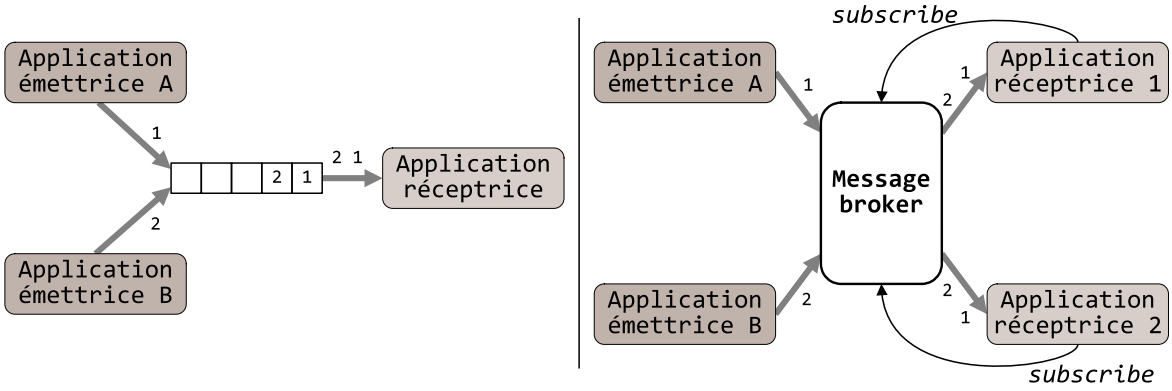


FIGURE 1.10 – Les deux classes de MOM : MOM basés sur file d'attente (gauche) et sur un courtier de messages (droite).

— Les MOM basés sur un mécanisme « Publish/Subscribe » ou mode par Publication/Abonnement : ce MOM Publish/Subscribe fournit une architecture dans laquelle les messages sont acheminés par un courtier en messages ou « Message broker ». Les récepteurs doivent s'abonner avant de recevoir des messages. Comme le montre la figure 1.10, la principale différence avec un MOM basé sur une file d'attente est l'architecture qui permet aux messages d'être acheminés via un courtier en messages et qui permet aux messages d'être reçus par tous les récepteurs abonnés.

Les MOM peuvent également inclure des fonctionnalités telles que la persistance et la réplication des messages et ils peuvent fournir des performances de qualité de service (QoS) [Capra 2001].

1.3.1.4 Middleware de type SOM / ESB

Le middleware de type SOM [Al-Jaroodi 2012] (Service Oriented Middleware) est une catégorie d'intergiciels basée sur une architecture orientée services (SOA). SOA (Service Oriented Architecture) est un modèle architectural de conception au sein duquel les composants d'application fournissent des services à d'autres composants via un protocole de communication sur un réseau.

Les principes sous-jacents à SOA sont indépendants de tout fournisseur, produit ou technologie. Un service doit mettre en œuvre au moins une action spécifique, comme par exemple la demande de la valeur d'un capteur, la mise à jour de la configuration d'une mission ou la modification des paramètres de l'environnement. Un middleware de type SOM est conçu comme un ensemble de services qui peuvent être utilisés pour faciliter le développement, l'exécution et la gestion des applications orientées services. Il s'insère comme les autres middlewares entre les principaux acteurs du SOC (Service Oriented Computing) à savoir les fournisseurs de services, les développeurs de services et les consommateurs de services.

SOM est ainsi composé de trois principaux composants : un fournisseur de services, un demandeur de services et un registre. Cette architecture permet aux fournisseurs de services de déployer facilement leurs composants et de publier leur présence dans le registre, afin que les

demandeurs de services puissent ensuite y accéder. Un mécanisme est par ailleurs mis en œuvre pour permettre la découverte des services publiés. SOM propose également une abstraction des services hétérogènes afin d'homogénéiser la manière d'y accéder.

Les communications peuvent être établies aussi bien en mode synchrone, qu'en mode asynchrone. La figure 1.11 représente une vue de l'architecture SOA propre à un middleware de type SOM, en partant des données, en passant par la couche d'abstraction des données, les services de données, les services, les couches d'orchestration et enfin la surveillance et la gestion des événements [Chitic 2018].

Les intergiciels SOA fournissent des mécanismes pour découvrir et localiser les services disponibles. Cela permet aux applications de trouver dynamiquement les services dont elles ont besoin Ils incluent généralement des mécanismes de gestion des politiques qui permettent de spécifier des règles et des contraintes concernant l'utilisation des services (authentification, autorisation, confidentialité ou qualité de service).

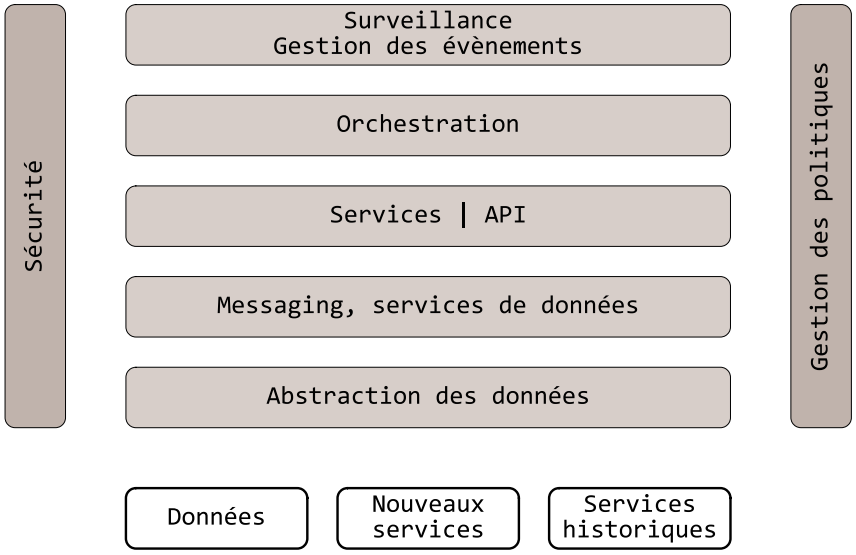


FIGURE 1.11 – L'architecture SOA qui fonde les intergiciels de type SOM et ESB d'après [Chitic 2018].

ESB est un autre type de middleware qui s'appuie sur l'architecture logicielle SOA pour la communication entre composants [Schmidt 2005]. Son modèle de communication est dérivé du modèle client-serveur et s'appuie sur un paradigme de bus logiciel. La principale différence entre ESB et SOM est l'absence de courtier central, ce qui rend l'ESB plus flexible et plus évolutif.

Comme le montre la figure 1.12 [Chitic 2018], ESB est capable de permettre à des composants hétérogènes de communiquer à l'aide de divers formats de stockage/transmission de données (marshalling) tels que JavaScript Object Notation (JSON) ou Simple Object Access Protocol (SOAP).

Les classes d'intergiciels présentées illustrent les architectures logicielles mises en œuvre afin de faciliter la distribution de composants logiciels sur des systèmes, des réseaux hétérogènes en

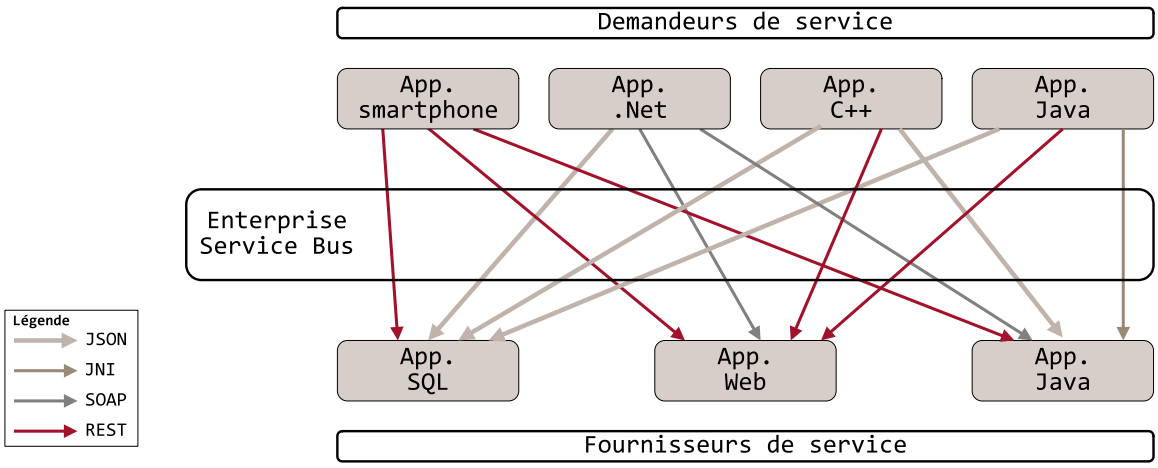


FIGURE 1.12 – Communication entre composants hétérogènes basée sur ESB.

permettant la communication et l'interaction entre ces composants.

1.3.2 A quels enjeux répondent les intergiciels de robotique ?

Les robots développés aujourd'hui en recherche intègrent une grande variété de composants matériels et logiciels développés dans des langages variés et reposant à la fois sur des standards et des protocoles de communication différents. Ces robots sont amenés à évoluer de plus en plus dans des environnements ouverts et dynamiques, à coordonner leurs actions avec d'autres robots, ce qui induit naturellement le développement d'applications logicielles avec des besoins de perception multimodale, partagée entre plusieurs robots, un traitement temps réel de la perception afin de produire en sortie une commande sûre et adaptée aux contraintes applicatives.

Pour faciliter le développement de ces applications logicielles, les intergiciels de robotique offrent une couche d'abstraction entre le système d'exploitation et l'application permettant à la fois de gérer cette forte hétérogénéité matérielle et logicielle et également ces besoins élevés d'interopérabilité.

Les intergiciels de robotique doivent ainsi composer avec les caractéristiques du robot d'une part, et les besoins propres à l'application d'autre part afin de répondre aux enjeux suivants [Mohamed 2009] :

- simplifier le développement des applications robotiques avec des API (Application Programming Interface) dédiées pour encourager la réutilisation et l'intégration ;
- accompagner la communication et l'interopérabilité entre composants et entre robots ;
- fournir des services et fonctions de haut niveau permettant de produire des tâches impliquant la collaboration entre robots ;
- fournir une couche d'abstraction logicielle permettant d'unifier et de simplifier l'accès aux composants afin de masquer la complexité propre à leur accès bas niveau et à leur hétérogénéité ;
- fournir des services permettant d'accéder en temps réel à d'autres systèmes (capteurs sans fils, serveurs, ...) ;
- proposer des services de base pour la robotique (planification, cartographie, saisie, localisation, ...) afin de proposer l'accès aux fonctions et algorithmes associés les plus communément utilisés ;
- offrir également des mécanismes d'auto-récupération, d'auto configuration et d'auto-adaptation afin de s'adapter à un environnement dynamique [Valle 2013].

La contrepartie d'avoir une couche d'abstraction masquant l'hétérogénéité des capteurs et des actionneurs est que le logiciel de commande est plus complexe. Il peut être aussi plus sujet aux défaillances, particulièrement dans le contexte de l'implémentation de schémas de contrôle multiaxe ou multirobot pouvant nécessiter un fonctionnement cyclique périodique avec des temps de cycle courts.

1.3.3 Synthèse des intergiciels en robotique

Les intergiciels de robotique ont largement été développés et étudiés dans la littérature [Chitic 2018], [Mohamed 2009], [Mohamed 2008], [Elkady 2012]. Ils sont parfois désignés sous les termes « framework », « middleware », « plateforme logicielle » dans la littérature. Il est difficile aujourd'hui d'appréhender les différences entre les nombreuses propositions d'intergiciels existantes, hormis le fait que certaines propositions sont clairement orientées vers certains domaines cibles, tels que la robotique mobile avec le contrôle de flottes de robots [Kramer 2007]. Pour essayer d'extraire les contributions les plus significatives du domaine nous nous sommes

appuyés sur les mêmes critères que [Chitic 2018] en basant notre synthèse sur les intergiciels les plus cités et sur la nécessité de pouvoir prendre en compte un environnement multirobot. Ces intergiciels dédiés à la robotique s'appuient pour certains sur les types de middleware introduits précédemment.

On recense dans la table 1.2, les middlewares les plus connus qui ont évolué pour certains depuis leur création. Pour compléter les éléments présentés dans ce tableau, on peut observer une grande variété de propositions pour répondre à ces enjeux relatifs aux développements d'applications logicielles robotiques. La plupart de ces propositions d'intergiciels ont émergé pour permettre aux laboratoires fondateurs ou entreprises de pouvoir uniformiser et accompagner le développement de leurs propres plateformes robotiques avec un souci de standardisation du code, de réutilisation et d'interopérabilité entre les différents composants mis en œuvre (capteurs, actionneurs, ...), avec également un besoin de simulation. On note également que les travaux relatifs aux intergiciels depuis quelques décennies n'ont pas conduit à de nouvelles propositions pertinentes depuis une dizaine d'années. En effet les intergiciels cités ont évolué avec des extensions, des travaux associés pour certains et l'adhésion d'une communauté a finalement permis d'assurer leur déploiement à une échelle plus large ; c'est le cas du middleware ROS qui est aujourd'hui devenu le middleware de référence dans le domaine de la robotique.

Dans [Chishiro 2009] les auteurs présentent l'extension OpenRTMaist pour RT-Middleware afin d'y intégrer des fonctionnalités de contrôle temps réel dur ; cette extension est souvent présentée à tort comme un middleware. De la même manière, Pyro ou Python Robotics [Blank 2005] est finalement une extension du middleware Player [Kranz 2006]. Il existe de nombreuses bibliothèques pour Pyro qui fournissent des services robotiques spécifiques déployés parallèlement sous Player. L'intergiciel est compatible avec les robots mobiles Pioneer, Sony Aibo et tous les robots supportés par Player/Stage. Le langage de programmation Python a contribué à son déploiement important dans le domaine de l'éducation.

On observe également que Orocos [Bruyninckx 2001], RT-Middleware [Ando 2005], Babel [Kwak 2006], IRSP [Fernandez 2006], Miro [Kruger 2006] s'appuient sur CORBA (Common Object Request Broker). OpenRTM-aist et OPRoS mettent en œuvre la norme de l'OMG pour standardiser la gestion des composants : le Robot Technology Component Standard de l'OMG. Pour autant un certain nombre d'intergiciels ont fait un choix différent, ce qui explique aussi probablement leur difficulté à s'imposer auprès d'une plus large communauté d'utilisateurs.

Un point important pour classer et comparer ces middlewares est de définir des critères permettant d'objectiver leurs performances et leurs capacités à répondre aux enjeux de la section 1.3.2. [Calisi] proposent une classification des middlewares existants (cf. table 1.3) s'appuyant en premier lieu sur la manière dont les tâches sont implémentées et sur la manière de partager des informations entre les tâches.

Une autre comparaison intéressante est proposée dans [Jang 2010] qui s'appuie sur un certain nombre de critères tels que les OS supportés, middleware de type RPC ou non, tolérance aux fautes, temps réel, cf. table 1.4. Les critères retenus valorisent OPRoS qui remplit tous les critères retenus, hormis un aspect essentiel relatif au temps réel. Les auteurs notent ainsi que OPRoS ne supporte pas l'ordonnancement temps réel et que l'exécution périodique de composants logiciels souffre de variations temporelles dues à l'ordonnancement non temps réel.

[Chitic 2018] choisit d'évaluer huit intergiciels de référence en s'appuyant sur trois groupes de critères : l'architecture, l'infrastructure et l'usage.

Architecture Comme illustré sur la figure 1.5, pour évaluer le premier groupe de critères propres à l'architecture, l'auteur considère trois sous-critères : *Overhead* qui correspond à la charge de calcul induite par le middleware, liée aux ressources logicielles nécessaires pour instancier le middleware avec les services associés ; *Vendor locking* correspond aux systèmes d'exploitation supportés ; et *Robustness to Failures*, qui considère la possibilité de redémarrer automatiquement le middleware en cas de défaut ou d'avoir un mode dégradé inhibant certains services. Avec cette grille de lecture, les middleware ROS et Player sont clairement avantagés.

Infrastructure Le second groupe de critères illustré sur la figure 1.6 s'intéresse aux caractéristiques techniques de l'intergiciel. Cinq sous-critères sont considérés : *MM* (Management and Monitoring) évalue la possibilité d'avoir une interface de gestion et de suivi de l'exécution des composants, services instanciés ; *MCS* (Multi-robot Coordination Services), la possibilité d'intégrer la coordination multirobot ; *SOTS* (Scheduled Operations and Tasks Services), la possibilité d'ordonnancer des tâches dans le middleware ; *DDSS* (Durable Data Storage Services), la possibilité de sauvegarder des informations propres aux capteurs, aux messages, et *COM* (COMmunication) considère la manière de communiquer entre les composants logiciels distribués.

Le sous-critère MM valorise ROS et MRDS car ROS intègre de manière native tous les outils nécessaires au développement et un tableau de suivi auquel il est possible d'accéder à distance et qui permet de monitorer tous les paramètres de l'application ; de même MRDS adossé à l'IDE Microsoft Visual Studio offre tous les outils de monitoring et débogage. Pour autant l'auteur aurait pu mettre au même niveau Player/Stage, MARIE et Pyro qui intègrent une interface graphique permettant de contrôler, visualiser les composants et leur état.

Coordination multirobot Le sous-critère MCS montre une des faiblesses de tous ces middlewares qui est de ne pas intégrer de manière native la coordination multi-robots ; en effet ROS, Miro, MRDS, Orca, MARIE et Pyro laissent l'application gérer cette coordination au niveau supérieur. Pour le sous-critère SOTS, la plupart des middlewares délèguent l'ordonnancement des tâches au système d'exploitation, et généralement à Linux qui reste l'OS le plus répandu ; pour MRDS il est géré par l'ordonnanceur de Windows. Seul le middleware MARIE intègre la possibilité en interne de gérer l'ordonnancement de tâches et services. Pour le sous-critère DDSS, seul le middleware ROS offre des services de stockage. Pour autant, il est possible avec Player/Stage, MRDS, Orca et Pyro de sauvegarder dans des fichiers texte, de même pour MIRO, MARIE et Carmen de sauvegarder dans des fichiers XML (Extensible Markup Language).

Le sous-critère COM est un critère essentiel. On distingue naturellement les communications synchrones et asynchrones, tous les middlewares ne permettent pas les communications synchrones en raison de leur conception interne. Pour Player/Stage, Pyro la méthode de communication repose sur des connexions directes via socket. Pour MARIE et MIRO, les échanges sont gérés via CORBA. Pour Carmen, elle est basée uniquement sur IPC. MRDS et ROS supportent à la fois la communication synchrone et asynchrone. MRDS utilise le protocole DSSP (Decentralized Software Services Protocol) basé sur SOAP et le protocole HTTP (Hypertext Transfer Protocol) pour établir les échanges entre les services. ROS prend en charge la communication synchrone via les services, la communication asynchrone via les topics, les messages structurés utilisant un langage de description d'interface (IDL) spécifique. ROS et MRDS supportent des protocoles standards comme TCP et HTTP.

Usage Pour terminer l'analyse de [Chitic 2018], l'auteur évoque un dernier groupe de critères relatifs à l'usage. Quatre sous-critères sont ainsi définis : *DLC* (Deployment and Life-Cycle) qui considère les outils utilisés pour faciliter, tester, accompagner le développement ; *PM* (Programming Model) considère le type de programmation (blocs séquentiels, sur évènements, ...) ; *CDIS* (Code and Data Integration Services) considère la possibilité d'intégrer de nouvelles fonctions, de nouveaux services via des API ; EPI (Extension Points and Interfaces) fait référence aux services très demandés. On observe que la plupart des middlewares satisfont au sous-critère DLC, car ils utilisent des outils de compilation standards tels CMake (catkin sous ROS s'appuie sur CMake) et ils offrent également la possibilité de se connecter à des environnements de simulation comme Gazebo. Naturellement Microsoft sort du lot avec son IDE Visual Studio qui offre toutes les fonctionnalités nécessaires sur l'ensemble des critères. De même pour la programmation, plusieurs langages sont supportés avec naturellement un avantage pour les langages plus proches du système tels C, C++, C# et plusieurs modèles de programmation synchrones, asynchrones également qui permettent de coder les comportements attendus sous formes de séquences, sur évènements ou avec des machines d'états. Pour le sous-critère CDIS, tous les middlewares offrent une architecture modulaire facilitant l'intégration et la réutilisation du code. De la même manière, ils offrent tous une variété de services liée généralement au domaine applicatif pour lequel ils ont été initialement développé. Pour autant ROS est aujourd'hui le grand gagnant, car il fédère la plus large communauté d'utilisateurs, qui contribue fortement au développement de nouveaux services dans tous les champs applicatifs de la robotique. L'analyse de ces trois grilles conduit ainsi l'auteur à mettre en avant ROS et en second lieu MRDS dans un contexte visé qui est celui de la mise en œuvre de flottes de robots.

De nombreuses autres grilles de lecture ont été proposées qui recoupent ces critères, par exemple [Mohamed 2008] ou [Mohamed 2009]. Les auteurs de ces différentes analyses convergent vers la même conclusion. Aucun des middlewares évoqués ne permet aujourd'hui d'adresser tous les enjeux et contraintes applicatives de tout système robotique.

Pour la plupart de ces middlewares, on peut relever les manques suivants qui sont préjudiciables dès lors que l'on souhaite contrôler de manière robuste plusieurs robots, machines multiaxes dans une cellule de production et plus généralement dans l'usine où évoluent en temps réel des robots, des humains, des machines potentiellement en interaction :

- les middlewares présentés sont orientés robotique et non orientés machine : les robots sont en effet amenés à être intégrés dans des cellules de production où leurs mouvements peuvent être coordonnés avec d'autres machines, portiques multiaxes ;
- les middlewares n'intègrent pas de manière native le déterminisme temporel de l'application du plus bas niveau au plus haut niveau : il devient alors difficile de garantir la réalisation de stratégies de contrôle mises en œuvre avec des temps de cycle préalablement définis pour chaque tâche ;
- les middlewares ne permettent pas d'assurer un haut niveau de coordination de plusieurs machines multiaxes, plusieurs robots dans le contexte de la réalisation d'une tâche conjointe : une telle coordination implique une communication temps réel dur entre l'ensemble des tâches qui peuvent être distribuées sur plusieurs processeurs à travers un réseau de machines, de robots coordonnés ;
- les standards industriels mis en œuvre par les fabricants de robots, de machines, par les spécialistes de l'automation ne sont pas pris en compte dans les middlewares présentés ce qui explique probablement en partie, la difficulté de transférer ces middlewares hors

des laboratoires de recherche ;
— l'intergiciel, par définition placé entre l'OS et le programme utilisateur, implique une rupture dans la continuité du développement applicatif.

Pour compléter cette analyse des middlewares existants, nous choisissons de présenter plus en détail trois middlewares qui nous semblent emblématiques parmi ceux cités précédemment, afin de mieux apprécier les caractéristiques inhérentes à ces intergiciels.

Table 1.2 – Principaux intergiciels de robotique.

Publ.	Intergiciel	Laboratoire	Architecture et caractéristiques
2001	Orocos [Bruyninckx 2001]	Université Catholique de Louvain, Belgique	Développement d'applications basées sur des composants de contrôle en temps réel configurables et interactifs. Largement utilisé dans la communauté robotique.
2003	CARMEN [Montemerlo 2003]	Carnegie Mellon University, USA	Ecrit en C, repose sur une architecture à 3 niveaux. Le 1er niveau fournit une abstraction matérielle pour les capteurs et autres composants. Le 2nd niveau offre des services (navigation, localisation, suivi d'objets, planification) du mouvement. Le 3ème niveau est l'application utilisateur et s'appuie sur les données provenant des couches inférieures et une communication IPC (Inter-Process Communication).
2004	MissionLab [Endo 2004]	Mobile Robot Laboratory, Georgia Tech, USA	Système de planification de mission basé sur une architecture de commande hybride délibérative et réactive pour les robots mobiles autonomes.
2004	Webots [Michel 2004]	Cyberbotics Ltd, spin off EPFL, Suisse	Environnement de prototypage pour modéliser, simuler et programmer un robot mobile.
2005	RT-Middleware [Ando 2005]	AIST, Tsukuba, Japon	Approche basée sur l'intégration de fonctions de systèmes robotiques par une gestion modulaire de composants logiciels basée sur CORBA.
2005	Orca [Brooks 2005]	Institut für Technische Informatik, Lübeck, Allemagne	Intergiciel open source pour le développement de systèmes à base de composants.
2006	Clarity [Nesnas 2006]	Jet Propulsion Laboratory, USA	Architecture embarquée de contrôle multicouche gérant l'exécution, la planification de tâches de navigation mobile autonome (application spatiale sur le Mars Rover).
2006	WURDE [Heckel 2006]	Washington University, USA	Architecture basée sur 4 niveaux d'abstractions : communications inspirées de CORBA, interfaces, applications et systèmes. Application en robotique mobile.
2006	Babel [Kwak 2006]	University of Malaga, Espagne	Système de développement composé d'outils pour l'implémentation, l'exécution, le débogage et la maintenance d'applications robotiques. Support de CORBA.

Publ.	Intergiciel	Laboratoire	Architecture et caractéristiques
2006	IRSP [Fernandez 2006]	Intelligent Robotics Lab, Séoul, Corée du Sud	Architecture basée sur CORBA avec 4 modules : créateur, assembleur, simulateur et exécuteur de composants fonctionne sous Linux.
2006	RIK [Bruemmer 2006]	Idaho National Laboratory, USA	Architecture à 4 niveaux : API, couche d'abstraction, comportements de robots réactifs et délibératifs, gestion asynchrone de comportements.
2006	YARP [Metta 2006]	LIRA-Lab, DIST, Univ. of Genova, Italie et MIT, USA	Support de communication interprocessus, traitement d'images, hiérarchie de classes et réutilisation du code sur plateformes matérielles variées. Orienté robotique humanoïde. Implémentation sous Windows, Linux et QNX6.
2006	Player [Kranz 2006]	Université de Munich, Allemagne	Les atouts de Player sont le serveur de devices, les langages de programmation supportés, le protocole de transport basé sur les sockets, la modularité et l'implémentation en open-source.
2006	Miro [Kruger 2006]	Chemnitz University of Technology, Allemagne	Architecture distribuée, orientée objet, destinée à favoriser l'intégration de logiciels hétérogènes, la modularité et la portabilité des applications robotiques. Basé sur CORBA.
2007	Marie [Côté 2007]	Université de Sherbrooke, Canada	Environnement logiciel intégré pour la robotique mobile et autonome. Créé en C++ Communication basée sur ACE (Adaptive Communication Environment) et sur un composant logiciel médiateur appelé Mediator Design Pattern (MDP) permettant aux composants logiciels de se connecter.
2008	OpenRDK [Calisi]	Universita di Roma, Italie	Architecture multi-processus associée à un partage de données et une communication inter-modules de type tableau noir. Implémentation sous Linux, OS X.
2008	MRDS [Johns 2008]	Microsoft, USA	Middleware sous Windows avec 4 composants : un runtime d'exécution (CCR), des services logiciels décentralisés (DSS), un langage de programmation visuel (VPL) et un langage de programmation visuel (VSE).
2009	SmartSoft [Schlegel 2012]	University of Applied Sciences, Ulm, Allemagne	Architecture basée modèle permettant de créer le squelette du code applicatif à partir du modèle.

Publ.	Intergiciel	Laboratoire	Architecture et caractéristiques
2009	OPROS [Jang 2010]	IT Convergence Technology Research Lab., Daejeon, Corée du Sud	La plateforme de composants de robots propose un modèle de composants, un outil de création de composants, et un moteur d'exécution de composants.
2010	Aseba [Magnenat 2011]	EPFL, Suisse	Architecture distribuée de contrôle basée évènements pour les robots complexes. Différence avec les autres middlewares : temps-réel dur et implémentation possible sur microcontrôleurs.
2010	OpenJAUS [Galluzzo 2010]	Département de la Défense, USA	Architecture pour les systèmes sans pilotes. Standard de communication ouvert pour soutenir l'interopérabilité des systèmes robotiques dans l'armée. Le SDK OpenJAUS est un ensemble de bibliothèques C/C++ qui fournit une interface de programmation abstraite (API).
2010	ORCCAD [Arias 2010]	Inria, France	Conception de contrôleurs de robots basée sur la théorie du contrôle et des évènements discrets.
2010	MRPT [Claraco 2010] [Rodríguez 2016]	Machine Perception & Intelligent Robotics Lab., Univ. of Malaga, Espagne	Librairie dédiée à la robotique mobile aujourd'hui intégrée à l'intergiciel ROS.
2010	ROS [Cousins 2010]	Stanford Artificial Intelligence Laboratory, USA	ROS est aujourd'hui le middleware de référence avec 3 points forts : la communication inter-composants, des outils intégrés pour le développeur, un grand nombre de librairies disponibles.

TABLE 1.3 – Résumé des différents frameworks de robotique, d'après [Calisi]

Intergiciel	Concurrence	Partage d'information	Outils	Usages
Orocos	Callbacks, threads	Ports de données sans verrouillage (CORBA)	Surveillance à distance, journalisation	Composants bas niveau
Orca	Processus	ICE	Surveillance à distance, journalisation	Robots mobiles
CARMEN	Processus	IPC	Journalisation, visualisation	Cartographie et navigation
OpenRTM	Threads	CORBA	GUI de configuration	Robotique générale
MRDS	Processus	HTTP/DSSP via DSS	Simulateur 3D	Robotique générale
Player	Serveur de threads	Client / serveurs, propriétaire sur TCP	Simulateurs 2D et 3D	Composants bas niveau
Clarity	Threads et processus	basé sur ACE	-	Spatial
MARIE	Processus	Basé sur ACE, ouverte sur protocoles tiers	GUI de configuration	Connexion entre frameworks
MIRO	Processus	CORBA	Journalisation	Robots mobiles
OpenRDK	Threads	Mémoire partagée, TCP / UDP propriétaire	Surveillance à distance, journalisation	Robots mobiles

TABLE 1.4 – Comparaison des plateformes logicielles pour la robotique, d'après [Jang 2010].

Intergic.	Open source	OS	Appel RPC	Env. simu.	Tol. fautes	Temps-réel
Orocos	Oui	Linux	Oui	Non	Non	Oui
MRDS	Non	Win	Oui	Oui	Non	Non
Player	Oui	Win Linux	Non	Oui	Non	Non
MARIE	Oui	Linux	Oui	Oui	Non	Non
MIRO	Oui	Linux	Oui	Non	Non	Non
RT-Middle.	Oui	Win Linux	Oui	Non	Non	Non
ROS	Oui	Linux	Oui	Oui	Non	Non
OPRoS	Oui	Win Linux	Oui	Oui	Oui	Non

TABLE 1.5 – Comparaison d'intergiciels de référence selon [Chitic 2018], partie 1 : architecture.

Intergiciel	Overhead	Vendor locking	Robustness to failures
Player	+	Linux Windows	+
ROS	+	Ubuntu Debian Windows MacOS	+
Miro	- CORBA	Linux	+
MRDS	- DDS et CCR	Windows	+
MARIE	- composants additionnels	Linux	+
Orca	- ICE	Linux	+
CARMEN	-	Red Hat SuSE	+
Pyro	-	Linux	-

Table 1.6 – Comparaison d'intergiciels de référence selon [Chitic 2018], partie 2 : infrastructure.

Intergiciel	Management & monitoring	Multi-robot coordination services	Scheduled operations and task services	Durable data storage services	Communication
Player	∼	+	∼	∼	∼
ROS	+	+	∼	+	+
Miro	-	+	∼	∼	∼
MRDS	+	+	∼	∼	+
MARIE	∼	+	+	∼	∼
Orca	-	+	∼	∼	∼
CARMEN	-	-	∼	∼	∼
Pyro	∼	+	∼	∼	∼

Table 1.7 – Comparaison d'intergiciels de référence selon [Chitic 2018], partie 3 : usages.

Intergiciel	Deployment and life cycle	Programming model	Code and data integration services	Extension points and interfaces
Player	∼	∼	+ déplacement des composants pendant l'exécution	+ accès par API
ROS	+ Catkin Gazebo	+ programmation synchrone et asynchrone	+ roslaunch rosrun	+
Miro	+ compilateur IDL Gazebo	- CORBA	∼	+
MRDS	+ Visual Studio	+ C#	+ VPL	+
MARIE	-	∼	+	+
Orca	+ CMake	+ plusieurs langages de programmation	∼	+
CARMEN	∼	∼	∼	+
Pyro	+ Gazebo	∼	+	+

1.4 Présentation de trois middlewares emblématiques

Les trois middlewares retenus Orocos, RT-Middleware et ROS rassemblent parmi les middlewares présentés précédemment une large communauté de chercheurs au niveau international et ils sont emblématiques dans leur conception et dans leur philosophie des enjeux associés à leur démarche de développement. Les fondateurs de ces middlewares montrent également la dynamique des acteurs internationaux impliqués sur ces sujets, en Europe pour Orocos, au Japon pour RT-Middleware, aux Etats-Unis pour ROS avec l'objectif de fédérer une large communauté au-delà des laboratoires fondateurs de ces middlewares.

1.4.1 Orocos

Le projet Orocos [Bruyninckx 2001], [Garcia 2013] est né en 2000 avec le principe original de développer un environnement logiciel gratuit permettant de contrôler les robots. La motivation de Herman Bruyninckx, à l'origine du projet, était de faciliter l'utilisation des logiciels de contrôle spécifiques à chaque robot. Le projet a été développé en partenariat avec trois partenaires académiques : la KU (Katholieke Universiteit) à Louvain en Belgique, le LAAS (Laboratoire d'Analyse et d'Architecture des Systèmes) à Toulouse en France et enfin l'Institut Royal de Technologie KTH (Kungliga Tekniska högskolan) à Stockholm en Suède.

Orocos a été développé pour offrir les caractéristiques suivantes :

— un environnement open source,
— un environnement flexible et modulaire,
— un environnement facile à prendre en main,
— un environnement indépendant des constructeurs de robot.

Orocos [Bruyninckx 2003] s'appuie sur le standard ORB, propre aux middlewares basés sur CORBA pour assurer la communication entre plusieurs composants logiciels. Pour construire une application robotique, Orocos s'appuie ainsi sur quatre bibliothèques C++ et une boîte à outils :

— le Real-Time Toolkit qui fournit l'infrastructure et les fonctionnalités pour construire une application robotique en C++ ;
— la librairie KDL (Kinematics and Dynamics Library) qui gère les calculs cinématiques et dynamiques sur des chaines cinématiques en temps réel,
— la librairie OCL (Orocos Component Library) qui fournit des composants de commande prêts à l'emploi,
— la librairie BFL (Bayesian Filtering Library) qui fournit des fonctions de filtrage comme les filtres de Kalman et les filtres à particules,
— La librairie rFSM (Reduced Finite State Machine), une implémentation de machines à états finis en langage de scripts Lua.

Une application robotique basée sur Orocos est ainsi construite en agrégeant des composants logiciels et en définissant les modes d'interaction de ces composants. La figure 1.13 illustre l'architecture d'un composant logiciel de la librairie OCL. Le composant résultant est instancié et exécuté via un moteur d'exécution. Le fonctionnement du moteur d'exécution s'appuie sur la lecture de messages gérés au sein d'une file d'attente et sur le statut de la tâche implémentée grâce à la classe `TaskContext`. Cette classe gère le composant logiciel sous forme de tâche depuis sa configuration basée sur la lecture d'un fichier XML, jusqu'à son exécution et son arrêt. La figure

1.13 résume les communications intercomposants qui permettent de construire une application basée sur Orocos en exploitant un modèle de communication utilisant CORBA.

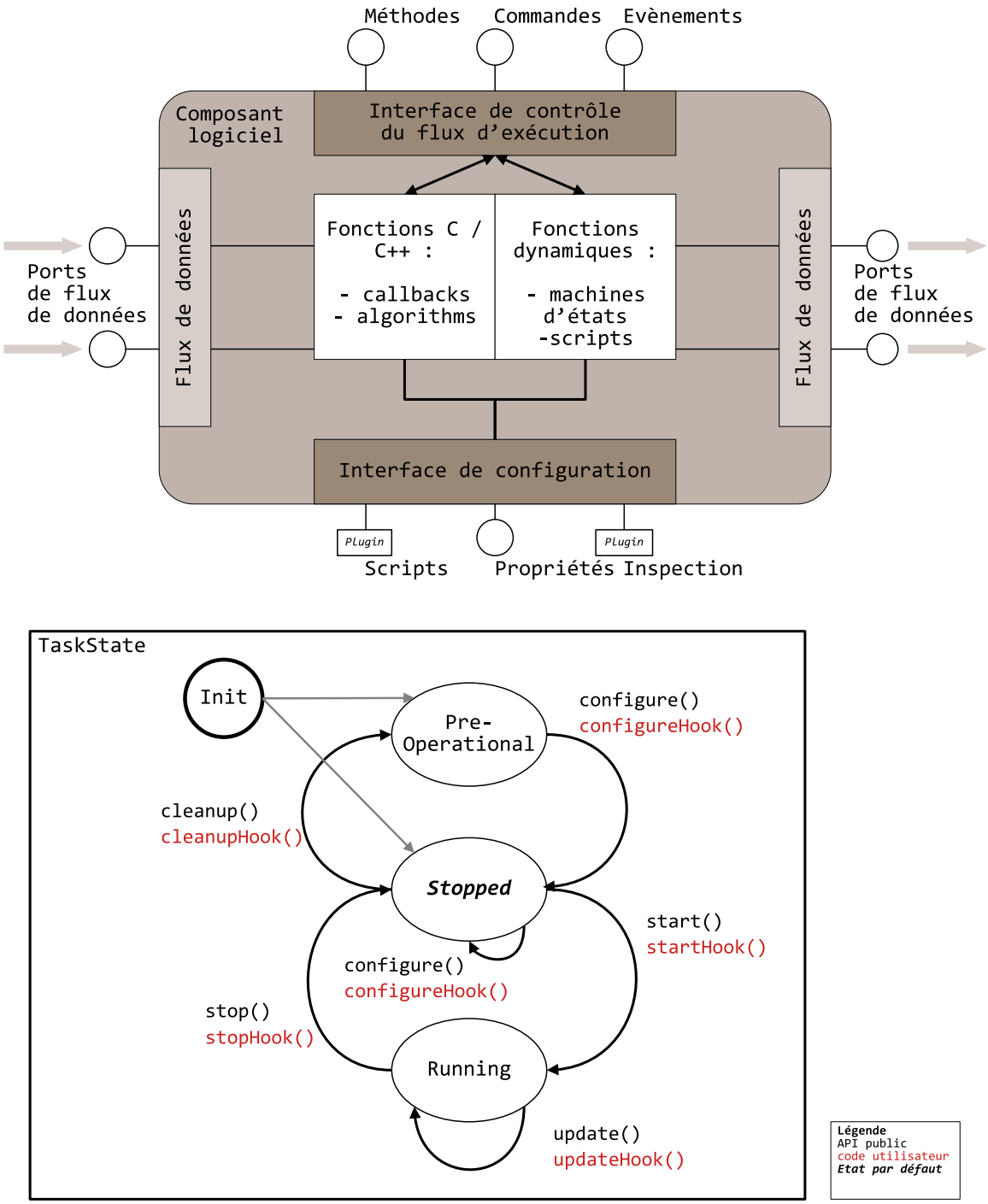


FIGURE 1.13 – Gestion de composants logiciels sous Orocos. Haut : architecture d'un composant logiciel, bas : classe `TaskContext`, gestion de la tâche associée au composant.

Un point fort d'Orocos est le fait qu'il s'appuie sur un standard de communication qui exploite CORBA, ce qui offre la possibilité de mettre en œuvre des communications synchrones entre les composants d'une application logicielle distribuée.

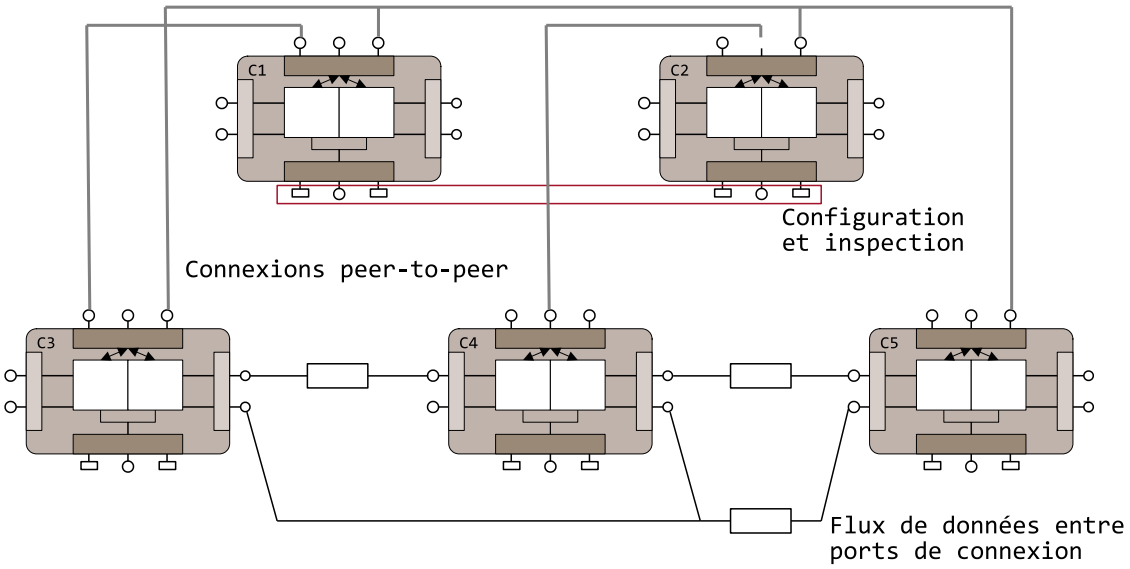


FIGURE 1.14 – Architecture d'une application robotique sous Orocos.

La construction d'une application temps réel distribuée sur plusieurs cibles est possible avec Orocos. Elle requiert d'implémenter Orocos en exploitant d'une part, l'extension Xenomai sous Linux pour obtenir des fonctionnalités temps réel dur et d'autre part, des bus de communication temps réel entre les cibles concernées. Pour autant, les seuls exemples proposés dans la littérature relative à Orocos sont basés sur une communication UDP ou TCP/IP, ce qui ne permet pas de garantir du déterminisme temporel.

Le projet Orocos était initialement financé en 2001 avec le soutien d'un projet européen. A la fin du projet en 2003, une version d'Orocos avec un contrôle simplifié de la position et de la vitesse d'un bras manipulateur composé de 6 degrés de liberté était disponible. L'Institut KTH a continué ce développement par la suite en développant ORCA. La différence entre ORCA et Orocos réside dans le fait qu'ORCA s'appuie sur le middleware ICE (Internet Communication Engine) [Henning 2004] en lieu et place de CORBA, jugé trop difficile à mettre en œuvre et moins ouvert en termes de programmation [Henning 2008]. Orocos a néanmoins continué à être utilisé et mis en œuvre pour accompagner le développement des applications logicielles robotiques du LAAS.

Il est important de noter qu'aujourd'hui Orocos peut être intégré à ROS, ce qui contribue à pérenniser les développements existants sous Orocos auprès d'une communauté plus large. En effet, les deux middlewares montrent un certain nombre de similitudes dans leurs concepts. Une couche de translation permet ainsi de faire le pont entre les concepts et caractéristiques associées propres aux deux middlewares comme illustré sur la table 1.8. Les dernières mises à jour à ce sujet datent de 2020.

Table 1.8 – Concepts communs entre Orocos et ROS.

Concept	Fonctionnalité Orocos	Fonctionnalité ROS
Communication par flux de données	Data ports	Topics
Terminal de données sortantes	Output port	Publisher
Terminal de données entrantes	Input port	Subscriber
Service	Operation	Service
Action	Action	Action
Structure de données	Typekit	Message
Variables partagées	Property, attribute, constant	Parameter
Entité de calcul	Orocos process, component	Node

1.4.2 RT-Middleware

RT-Middleware (RT pour *Robot Technology*) [84] est issu du projet NEDO (New Energy and industrial technology Development Organisation) initié en 2002 avec le soutien de l'industriel Matsushita et de la JARA (JApan Robot Association). Dans ce contexte, il visait à adresser aussi bien une utilisation industrielle, qu'une utilisation plus académique.

Comme Orocos, il s'appuie sur le modèle CORBA. RT-Middleware est d'abord une plateforme commune pour les robots, basée sur la technologie des objets distribués. Le développement d'une application robotique s'appuie ainsi sur la création, la gestion de composants logiciels en interaction nommés RT-components. Les objectifs de RT-middleware restent identiques à ses concurrents, il s'agissait de démocratiser l'utilisation de robots grâce à des développements efficaces et facilités. Sa force réside dans sa modularité et sa capacité à être utilisé sur différentes plateformes.

Pour ce faire, RT-Middleware s'appuie sur une architecture hiérarchisée à trois niveaux pour la gestion des composants matériels. Le plus bas niveau concerne les moteurs, les capteurs et les contrôleurs. Le niveau intermédiaire concerne les systèmes de vision et les algorithmes de traitement associés, les robots manipulateurs et les robots mobiles. Le niveau le plus élevé concerne une cellule complète telle que par exemple, un robot humanoïde avec ses capteurs proprioceptifs et un système de vision.

Au sein d'RT-Middleware, la gestion des éléments robotiques, tels que les actionneurs, est affectée à des composants nommés RT-component, et l'ensemble de la gestion du système robotique est construite en connectant ces composants. Cette architecture distribuée aide les développeurs à réutiliser les éléments robotiques et ainsi à augmenter la fiabilité du système robotique avec des objets de référence pour chaque composant (actionneur, capteur, robot...).

Le cœur de RT-Middleware réside dans l'architecture du composant RT-Component illustrée sur la figure 1.15. On note que chaque composant `RT-Component` a un port entrées-sorties pour communiquer avec les autres composants. Chaque port a un type spécifique et les ports qui ont le

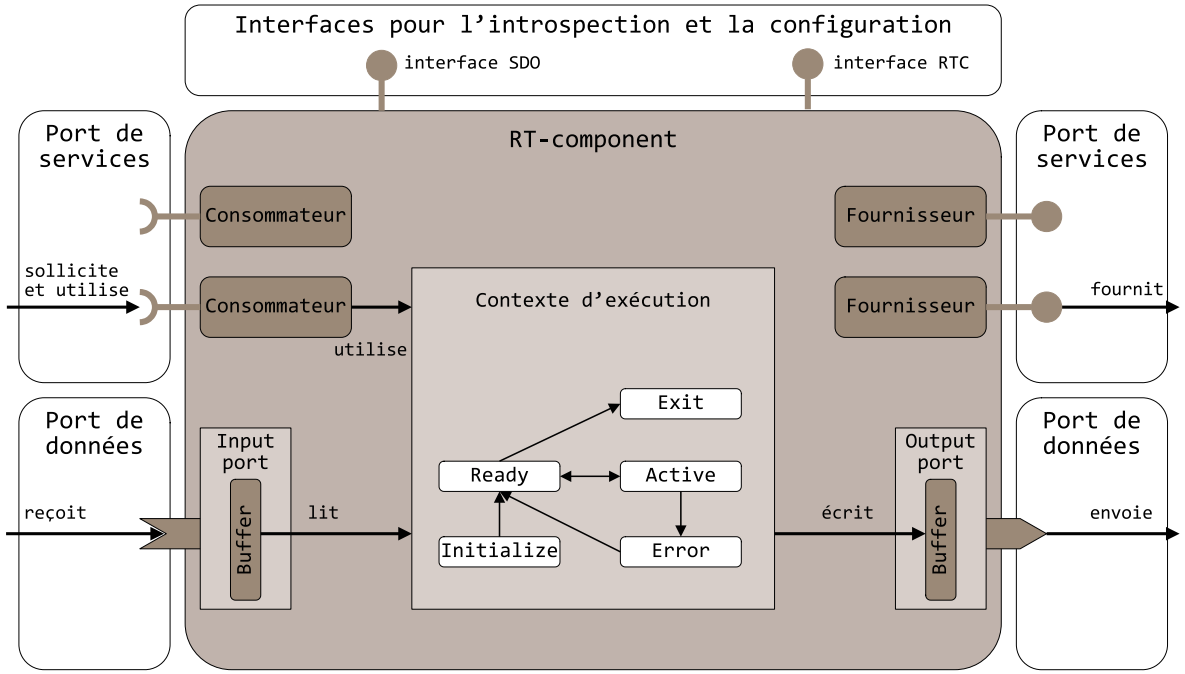


FIGURE 1.15 – Le composant `RT-Component` sous RT-Middleware.

même type peuvent être connectés entre eux. Le fonctionnement d'un composant RT-Component est également géré par une machine d'état interne. On note ainsi les états du composant suivants : `CREATED`, `INACTIVE`, `ACTIVE` et `ERROR`. Ces états et le comportement du composant sont contrôlés à chaque instant par le contexte d'exécution propre à chaque développement robotique.

La dernière extension de RT-Middleware est OpenRTM-aist. Il s'agit d'une plate-forme logicielle développée sur la base du standard RT-middleware. OpenRTM-aist a été développée par le National Institute of Advanced Industrial Science and Technology qui a contribué également à la définition du standard RT-middleware au Japon. La dernière version stable disponible écrite dans les langages de programmation C++, Java, Python a été mise à jour en 2019. Elle est disponible sous Windows, Linux, macOS, VxWorks, TOPPERS(ITRON), QNX.

1.4.3 ROS

Le middleware ROS est aujourd'hui le middleware qui fédère la plus large communauté au niveau national et international. Il se décline en deux versions ROS 1.0 et ROS 2.0.

1.4.3.1 ROS 1.0

ROS est l'acronyme de *Robot Operating System* ; il a été initialement développé dès 2007 par Stanford Artificial Intelligence Laboratory dans le cadre du projet Stanford AI Robot STAIR. A partir de 2008, la société américaine Willow Garage poursuit le développement de ROS pour accompagner le développement du robot mobile collaboratif PR2 [104]. Le succès de ROS, midd-

leware open source et gratuit est directement lié à la communauté importante de contributeurs qui l'utilisent et contribuent à son développement.

Comme évoqué précédemment, ROS offre quatre points clés à l'origine de son succès :

— la communication inter-composants, c'est-à-dire la possibilité de connecter de nombreux composants logiciels en exploitant des nœuds, quelle que soit la répartition des nœuds sur le réseau ;

— les outils de développement : ROS offre de nombreux outils d'analyse, de débogage, et d'affichage pour construire une application robotique (l'interface graphique 3D rviz pour afficher un environnement robotique, une carte de navigation, l'interface graphique `rqt_graph` pour analyser le graphe d'applications et les flots de données, `rosbag` pour rejouer des séquences topics, rqt une interface de contrôle incrémentale, ...) ;

— les librairies disponibles : ROS propose aussi bien des librairies pour la vision, que pour la construction d'environnements, la planification de mouvements, la planification de tâches ou le contrôle ;

— la communauté très importante et grandissante des utilisateurs et contributeurs.

ROS s'appuie sur trois concepts fondateurs illustrés sur la figure 1.16 :

— les Nodes ou nœuds : un node est un exécutable qui utilise ROS pour communiquer avec d'autres nodes ;

— les Topics : les nodes peuvent publier des messages sur un topic, aussi bien que souscrire à un topic pour recevoir des messages ;

— ROSMaster : c'est le processus cœur, il comprend un serveur de noms et permet de mettre les différents nœuds en contact pour qu'ils puissent communiquer.

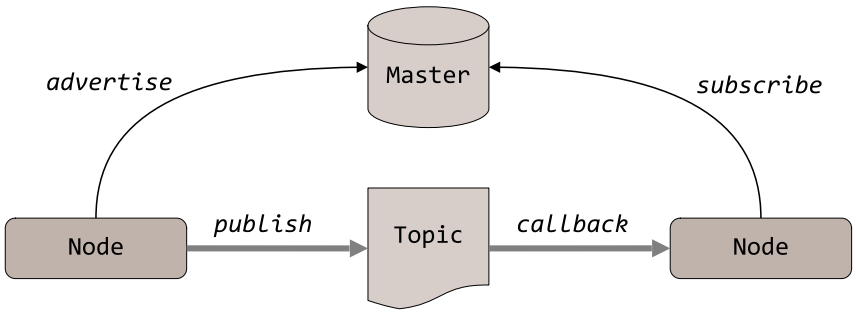


Figure 1.16 – Schéma de fonctionnement de ROSMaster.

Si on fait l'analogie avec les middlewares de type ORB (section 1.3.1.2), ROSMaster joue le rôle de broker ; de ce fait, la configuration réseau est simple et ne nécessite pas de spécifier une adresse IP ou un port pour faire interagir deux nœuds ensemble.

Pour illustrer l'implémentation des nœuds qui implémentent les programmes dans une application robotique, on peut citer l'exemple du pilotage d'un robot mobile : un nœud pourra être affecté à la perception avec la gestion du scrutateur laser, un nœud pourra assurer le contrôle des moteurs, un nœud pourra assurer la reconstruction de l'environnement...

La communication entre les différents nœuds peut être réalisée à l'aide de messages et de topics de manières asynchrone. Un message est simplement une structure et les types standards

de variables sont supportés (float, integer, boolean, ...). La communication se fait alors sur le principe publish/subscribe comme pour les middlewares de type MOM (section 1.3.1.3). Un nœud qui partage l'information la publie sur un topic, puis si un nœud est intéressé par cette dernière, alors il s'abonne à ce topic. Plusieurs nœuds peuvent publier ou s'abonner sur un même topic. Les nœuds publiant l'information ne savent pas à quel nœud bénéficieront ces données.

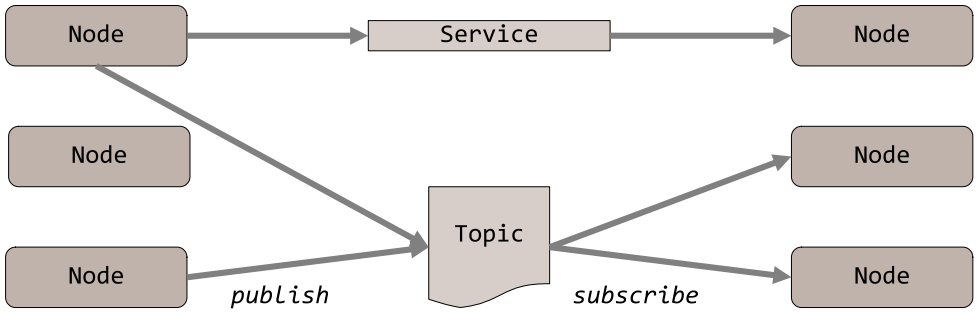


FIGURE 1.17 – Communication entre nœuds sous ROS : services et topics.

Un autre moyen de communiquer entre deux nœuds s'appuie sur les services, ce qui permet d'installer une communication synchrone entre nœuds. Contrairement aux topics qui sont un moyen de communication flexible, les services fonctionnent sur un principe de question/réponse et permettent d'échanger des données uniquement entre deux nœuds. La figure 1.17 illustre ces principes propres à la communication sous ROS.

1.4.4 ROS 2.0

ROS 2.0 [KAY et al., 2016] a été développé depuis 2007 pour répondre à certaines limites de ROS 1.0, concernant notamment :

- la mise en oeuvre simultanée de plusieurs robots en s'appuyant sur des standards ;
- la possibilité de gérer un fonctionnement temps réel dur inter-processus et inter-machines ;
- la robustesse et la qualité des distributions ou comment passer du développement en laboratoire à des applications dans le monde réel.

Pour répondre à ces nouvelles contraintes, ROS 2.0 a totalement modifié sa manière de communiquer. ROS 2.0 s'appuie ainsi sur le middleware DDS (Data Distribution Service) ; DDS permet d'assurer le transport sur un principe publish/subscribe similaire à ROS 1.0. Toutefois ce standard géré par l'OMG (Object Management Group) est évolutif, temps réel fiable et interopérable. Il est en l'occurrence utilisé sur Android [Computing 2012]. Avec ces modifications importantes au niveau de la communication, il était difficile pour les développeurs de faire évoluer simplement ROS 1.0 vers ROS 2.0 en conservant la compatibilité ascendante en raison des modifications structurelles requises. ROS 2.0 n'est donc pas compatible avec ROS 1.0. Pour terminer cette présentation de ces trois middlewares de référence nous avons établi le tableau de la figure 1.9 qui reprend leurs principales caractéristiques.

Table 1.9 – Tableau de synthèse, comparaison des principaux intergiciels.

Intergiciel	Techno. com.	Temps-réel dur	Installation et développement	Structure du projet
RT-Middleware	CORBA	non	Nécessite RTC-Daemon pour la gestion des composants RTC, programmation en C++, Java ou Python	Trois couches : 1 moteurs, capteurs 2 vision, traitement d'image 3 modélisation robot
Orocos	CORBA	oui (si Xeno-mai)	Orogen (langage pour la gestion des composants). Nécessite RTT pour la gestion des composants avec langage de script	Trois bibliothèques (composants, cinématique et dynamique, filtres bayésiens).
ROS	MOM RPC	non	Nécessite ROS-Run pour la gestion des nodes et topics. Programmation C++, Java, Python	12 versions, 7 versions non maintenues, 1600 packages et modules (robots, périph., capteurs, planification)
ROS 2.0	MOM RPC	oui (si Xeno-mai)	Nécessite ROS-Run pour la gestion des nodes et topics. Programmation C++, Java, Python.	Nombre de packages limité.

1.4.5 Limites de la communication basée sur ROS pour le contrôle multirobot

Pour valider notre choix de développer une méthodologie de développement alternative à l'usage d'un middleware nous avons évalué les performances de l'intergiciel ROS dans le contexte des travaux menés dans l'équipe RoBioSS : le contrôle-commande multiaxe étendu au contrôle multirobot [Fischer 2019].

Dans de nombreuses applications robotiques, ROS est généralement implémenté sur des PC standards avec des systèmes d'exploitation non temps réel. Il nous a semblé utile de chercher les limites de cette architecture. La version testée du middleware était ROS INDIGO sous le système d'exploitation Linux Ubuntu 14.04 sur un PC Intel Quad Core i7 (CPU M620 2.7GHz, 4Go RAM). Pour évaluer la faisabilité du contrôle dynamique multi-robots avec ROS, nous avons conçu un dispositif expérimental comprenant deux robots et des réseaux à haut débit basés sur Ethernet. Un haut niveau de coordination est nécessaire pour assurer le déterminisme temporel et la robustesse d'une cellule multirobot. Pour notre évaluation, nous avons utilisé les dispositifs suivants (figure 1.18) :

— Composant 1 : un PC industriel (B&R Automation PC910) qui contrôle un robot Comau Racer 3 et ses périphériques (préhenseur et capteurs) via le bus de terrain POWERLINK ;
— Composant 2 : un PLC industriel (B&R Automation X20CP1586) qui contrôle un robot Kuka iiwa équipé d'un bus de terrain EtherCAT.

L'objectif du test était de connecter ces deux composants complexes via l'intergiciel ROS afin

qu'ils partagent leurs états respectifs à une fréquence adaptée à la coopération : des valeurs objectif de $2ms$ et $20ms$ ont été choisies.

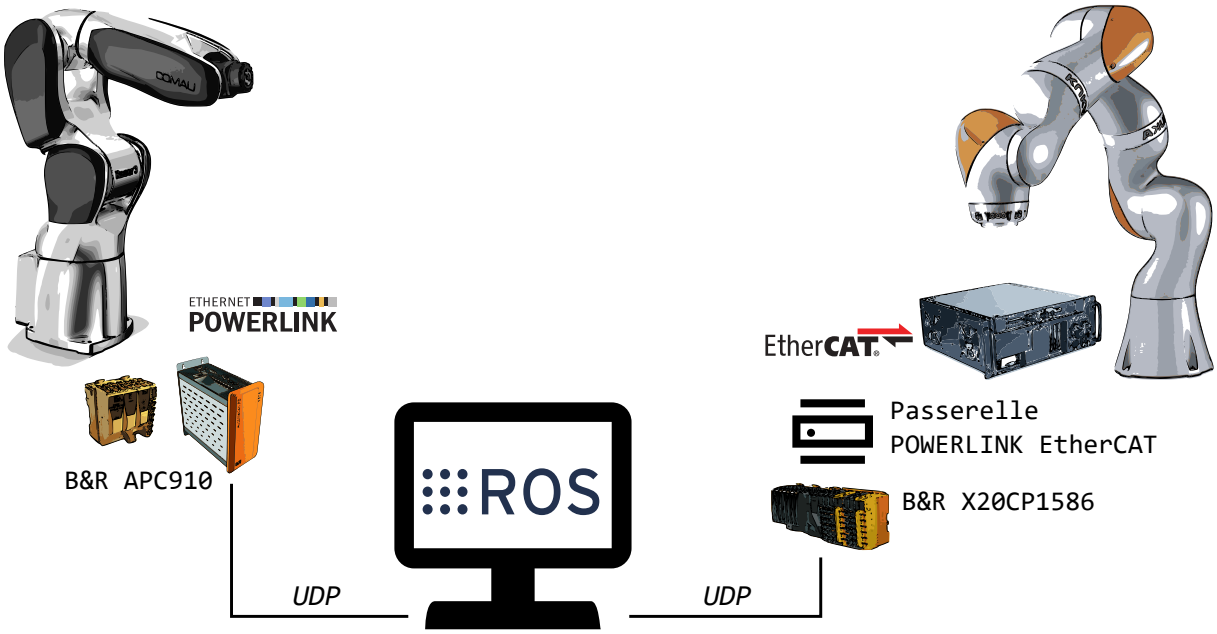


FIGURE 1.18 – Architecture utilisée pour l'évaluation de la communication ROS.

Communication basée sur ROS Pour mesurer la cohérence de la communication, nous avons choisi une application échangeant des données à un seul nombre entier : le composant 2 a besoin de connaître en permanence l'état d'un capteur associé au composant 1. Le composant 1 envoie périodiquement les données d'un capteur au composant 2 par le biais de nœuds et de topics ROS. Le composant 2 reçoit les données du capteur (points rouges sur la figure) et le temps de transfert est évalué. Les essais menés avec ROS ont révélé le fait que la communication entre les deux composants n'est pas déterministe quelle que soit la période. De plus, certaines données sont perdues si la fréquence d'échange est élevée.

Pour évaluer ROS2, nous avons utilisé la même architecture : un nœud par composant. Dans notre cas, le premier nœud reçoit des données et les publie sur le topic. Le second s'abonne au sujet et envoie des données via la communication UDP. A une période de $2ms$, aucun message n'est perdu mais certains échanges présentent une période de plusieurs périodes.

Ces essais, réalisés en 2019 au moment de la mise en service de ROS2, ne sont certainement plus représentatifs du niveau de performance actuel de l'intergiciel. [Park 2020] a, par exemple, évalué les améliorations obtenues par le passage de ROS à ROS2 installés sur des distributions Linux avec le patch Preempt RT, qui modifie le système d'ordonnancement des tâches du système d'exploitation. Les auteurs mesurent une faible latence de nœuds ROS2 exécutés périodiquement et un respect de la priorité des tâches compatible avec leur application. Cependant, le développement soutenu du framework logiciel *ros2_control* montre qu'il est nécessaire d'apporter beaucoup d'attention au développement de tâches de contrôle-commande pour lesquelles les exigences de comportement temps-réel sont essentielles. Plus récemment, [Varillon 2021] propose

la création d'une bilbiothèque complémentaire nommée *Corail* qui attribue aux tâches ROS2 un comportement déterministe. Cette bibliothèque prend en charge la création des fils d'exécution (threads), leur synchronisation et le respect des priorités entre tâches.

L'intergiciel ROS2 est aujourd'hui l'intergiciel robotique de référence, il est adapté à un grand nombre d'applications robotiques en proposant un très grand nombre de services, aussi bien en termes de composants matériels compatibles, qu'en termes d'algorithmes disponibles. Cependant, le cahier des charges retenu pour l'environnement logiciel de nos dispositifs, présenté dans le chapitre suivant, nous a conduit à choisir une stratégie différente.

1.5 Vers un environnement logiciel intégré de contrôle multiaxe

Comme pour ROS, le succès d'une nouvelle proposition d'environnement logiciel pour le contrôle en robotique dépendra de sa capacité à pouvoir le transférer hors du laboratoire auprès d'une large communauté. Plus précisément nous souhaitons, contribuer au développement d'applications de contrôle de machines multiaxes.

A la différence des intergiciels évoqués, nous préférerons utiliser le terme d'environnement logiciel intégré pour le contrôle multiaxe. L'idée centrale est de placer le contrôle-commande robotique au plus près des contrôleurs de mouvement, sur des cibles dont le comportement déterministe est garanti.

Dans [Gerkey 2002], les auteurs développent MURDOCH, une nouvelle méthode d'allocation dynamique des tâches basée sur le principe « publish/subscribe » déjà évoqué, et ils optimisent le temps d'exécution des tâches. Dans [Sandholm 1993] et [Ramos 1996], l'importance de la gestion des contraintes de temps avec l'utilisation du protocole Contract Net est soulignée. Nous pensons comme les auteurs qu'il est nécessaire de garantir le temps d'achèvement et une communication robuste pour obtenir une coopération entre des robots opérant dans des environnements dynamiques. A titre d'exemple, une méthode d'allocation dynamique des tâches pour des groupes de robots [Parker 1998] ne sera pas assez efficace pour assurer un haut niveau de synchronisation et un comportement déterministe si les robots ne mettent pas en œuvre un échange de données en temps réel ou si leurs contrôleurs ne donnent pas un accès en temps réel aux boucles de contrôle de bas niveau (boucles de contrôle de position, de vitesse ou de couple). L'objectif de notre proposition est d'unifier l'accès aux ressources des robots à partir du contrôleur industriel de la cellule robotique.

Actuellement, la programmation des machines industrielles repose sur la norme IEC 61131 de programmation des automates qui est une alternative à l'utilisation extensive de langages propriétaires pour chaque marque de robot. L'efficacité et la robustesse des logiciels de contrôle des machines et des processus sont également accrues par une communication en temps réel entre les contrôleurs de mouvement et les contrôleurs de robot. Malheureusement, le protocole Ethernet temps réel dur n'est pas une option disponible sur la plupart des armoires de commande des principaux fabricants de robots industriels.

Il s'agit ainsi de trouver un compromis entre la performance, la fonctionnalité et la standardisation pour avoir une solution appropriée. Il existe trois types de stratégies de développement. La première consiste à développer des programmes étroitement couplés au matériel. Grâce à cela, de hautes performances sont attendues, mais le programme ne peut pas être réutilisé, donc le coût et le temps du prochain développement ne seront pas réduits. La seconde consiste à développer de nombreuses fonctionnalités, ce qui est utile mais au détriment de la performance ; c'est

le cas des intergiciels existants. La dernière se concentre sur la standardisation et la réutilisation des composants du projet.

Une particularité de notre proposition est de ne pas dissocier les deux mondes que sont la robotique et l'automation ; le développement des machines multiaxes et celui des robots peuvent être abordés avec la même approche, d'autant plus qu'ils sont souvent associés, couplés au sein d'une même cellule de production. Notre proposition vise à proposer une nouvelle approche du développement des cellules de production automatisées associant robots et machines multiaxes.

Pour répondre aux enjeux et contraintes évoqués, nous choisirons d'appuyer le développement de notre solution logicielle de contrôle multiaxe sur les caractéristiques suivantes :

- Respect et extension du standard IEC 61131. Les langages largement disponibles du standard IEC 61131-3 peuvent être complétés avec des tâches et des bibliothèques réalisées en C++ pou réaliser des composants logiciels réutilisables ;
- Respect du standard PLCopen Motion Control : ce standard fournit toutes les fonctions monoaxes utiles et permet d'utiliser des configurations matérielles de n'importe quel fabricant du monde de l'automation (Siemens, Beckhoff, Rockwell, Schneider, B&R ...) avec également une forte implication des fabricants de robots (ABB, Stäubli, Yaskawa) ;
- Usage des protocoles de communication temps réel dur [EPSG 2016] : POWERLINK IEEE 61158 et EtherCAT répondent tous deux à ces contraintes. L'utilisation de protocoles Ethernet industriels est la clé de voûte de notre proposition temps-réel. Par exemple, le standard open source POWERLNK établit une communication en temps réel, avec une période ajustable, entre les composants d'automatisation ou les CPU et il respecte le standard Ethernet IEEE 802.3 sans accès modifié au média Ethernet.
- Une implémentation possible sur une variété de plateformes :
  - des machines de type PC sous Linux ou Windows,
  - des automates industriels ,
  - des microcontrôleurs.
- Une architecture avec une distribution modulable des composants logiciels et des boucles de contrôle de bas niveau associées en fonction des ressources matérielles disponibles.
- Un mode de simulation intégré permettant de simuler le comportement réel de l'application avant exécution.
- Un lien haute-fidélité entre machine réelle et machine virtuelle exploitant la simulation intégrée.
- La possibilité de s'interfacer facilement avec l'environnement logiciel intégré de contrôle multiaxe : cela permettra par exemple de développer des packages sous ROS permettant d'accéder aux fonctionnalités de l'application robotique développée avec notre environnement.

Avec la partie 1, nous souhaitions positionner le cadre de nos travaux dans un contexte où de nombreux développements ont été conduits, aussi bien sur les architectures de commande d'une manière générale, que dans le contexte des intergiciels propres au domaine de la robotique.

Nous avons pu démontrer à travers l'état de l'art proposé sur ces sujets, qu'un certain nombre de points ne sont pas adressés aujourd'hui par les propositions existantes. L'ouverture des robots dans un environnement de production autrefois cloisonné, avec des contraintes d'interaction avec d'autres machines actionnées et l'humain a modifié ces dernières années, les enjeux relatifs à l'accompagnement de la mutation de l'outil production. Les besoins de flexibilité, d'agilité accrus associés à la sûreté de fonctionnement et la sécurisation de l'interaction nous ont amené

à proposer un nouvel environnement logiciel intégré de contrôle multiaxes que nous présentons dans la suite de ce mémoire.

Le chapitre suivant introduit la bibliothèque *rtrmac* qui définit le modèle de composants logiciels que nous utilisons et les briques logicielles élémentaires qu'elle fournit. Je présente ensuite la démarche de création de nouveaux composants sur des exemples qui seront nécessaires pour la création de logiciels de contrôle-commande de systèmes mécatroniques.

Le troisième chapitre décrit l'extension de la bibliothèque au pilotage multiaxe de mécanismes et de robots.

Enfin, le dernier chapitre illustre l'adaptabilité de la méthode que nous avons développée sur des exemples de réalisation. Ceux-ci vont de l'informatique embarquée pour le contrôle d'un dispositif mécatronique, à l'association étroite du contrôle-commande à un jumeau numérique pour proposer une solution complète qui couvre plusieurs étapes du cycle de vie du logiciel.

Chapitre 2

Vers un système de contrôle-commande unifié

Le domaine d'intérêt de notre étude se limite à la conception de systèmes cyber physiques partageant les caractéristiques suivantes :

- le système doit présenter un comportement déterministe en contrôlant un nombre important de sorties et en réagissant à de nombreuses entrées,
- il doit pouvoir s'intégrer dans des ensembles plus vastes, en coopérant avec d'autres systèmes ou avec des opérateurs humains et en s'intégrant dans un système d'information (supervision, gestion de ligne de production, d'usine),
- le système doit pouvoir réagir à des perturbations extérieures et interagir avec un environnement dont l'évolution doit être prise en compte,
- le système est composé de constituants électroniques et électrotechniques intégrés dans des produits industriels répondant aux exigences des standards du domaine de l'automation industrielle.

Nous présentons dans un premier temps les exigences des standards nécessaires à la mise en œuvre de notre solution de développement de systèmes de contrôle commande. Puis nous décrivons les principes de la bibliothèque logicielle qui structure nos projets logiciels. Pour terminer, je présente la méthodologie associée à cette bibliothèque. Pour cela, je crée de nouveaux d'objets logiciels qui sont utiles pour le contrôle-commande robotique présenté dans le chapitre trois.

2.1 État des lieux des systèmes de contrôle-commande de machines

Le choix de se placer dans le cadre d'usage des technologies de l'automation industrielle couvre un ensemble de secteurs applicatifs qui dépasse le cadre initial des machines de production de l'industrie manufacturière. En effet, on retrouve ces composants (automates programmables industriels, cartes d'entrées-sorties, contrôleurs de mouvement) dans les domaines suivants (la liste n'est pas exhaustive) :

- la production et la gestion de l'énergie,
- l'industrie navale,
- la robotique agricole, les engins de chantier et de travaux publiques,
- les parcs d'attraction,
- l'automatisation des bâtiments,
- les bancs de tests et équipements scientifiques,
- les équipements médicaux.

Toutes ces applications partagent les exigences de durabilité et de robustesse des équipements, de comportement déterministe des systèmes et donc des applications logicielles, des exigences

de traçabilité et d'identification des responsabilités. Ces applications partagent aussi les exigences de qualification des acteurs de la mise en œuvre des systèmes automatisés produits, en particulier dans l'installation, la conduite, la maintenance et l'amélioration des équipements. Ce dernier aspect est également un point clé dans notre choix d'exploiter les standards du monde de l'automatisme.

2.1.1 Un écosystème vaste et en pleine transition

2.1.1.1 Des solutions technologies éprouvées, unifiées par des standards

Les systèmes de contrôle-commande temps réels industriels sont construits à partir de composants matériels produits par un très grand nombre de fournisseurs. Ce secteur est très dynamique et est constitué de très grosses multinationales mais aussi d'un grand nombre d'entreprises de taille intermédiaire. A titre d'exemple, la société CODESYS group qui réalise une suite logicielle de programmation en automatismes indépendante des cibles matérielles revendique plus de cinq cents entreprises qui utilisent leurs produits.

Dans le cadre de cet environnement très concurrentiel, même si ces équipements font appel à des solutions technologiques différentes, ils partagent des propriétés communes, en particulier des exigences de fiabilité et de comportement déterministe, l'implémentation d'un modèle logiciel et la mise à disposition d'un ensemble de langages de programmation. Ceci permet à leurs clients qui réalisent des composants, des éléments de machine ou des lignes de production d'avoir un très large choix de fournisseurs de matériels tout en utilisant des environnements de développements différents.

Cette situation résulte d'un très fructueux travail de standardisation porté par l'association internationale PLCopen qui s'est traduit par la publication de la norme IEC 61131 en 1993. Cette norme qui définit les spécifications pour les systèmes d'automatisation industrielle et les logiciels de programmation pour les automates programmables industriels (en anglais, PLC : Programmable Logical Controllers) a été révisée et mise à jour plusieurs fois pour répondre aux exigences de l'industrie et aux évolutions technologiques. Sa dernière version est la version IEC 61131-3 :2013.

Aujourd'hui, les travaux de standardisation de PLCopen se poursuivent sur d'autres aspects, dont plusieurs sont détaillés dans la suite de ce document. Le monde de l'automation évolue rapidement avec l'apparition de nombreux nouveaux acteurs et de nouvelles perspectives et ambitions. Cette tendance est liée à l'adoption et à la mise en œuvre des concepts d'Industrie 4.0 ou d'Usine du Futur, qui sont en train de provoquer des changements technologiques, méthodologies et organisationnels très importants.

2.1.1.2 Un univers en pleine transformation

L'« Industrie 4.0 » est une nouvelle étape de l'évolution industrielle qui est portée par la possibilité d'intégrer les dernières avancées technologiques et méthodologiques des Technologies de l'Information. Ce concept a été initialement développé par le gouvernement allemand dans l'objectif d'assurer le développement de son industrie manufacturière. La réflexion sur cette initiative stratégique a été lancée en novembre 2011 à horizon 2020. La version finale du rapport « Recommendations for implementing the strategic initiative INDUSTRIE 4.0 » [Kagermann 2013] porté par le ministère fédéral de l'éducation et de la recherche a été publiée en avril 2013. Bien

que ce document soit sous-titré « Securing the future of German manufacturing industry », ce concept a été adopté par le secteur industriel mondial.

Dix ans plus tard, ce travail de prospective est en train de se mettre en place et provoque des modifications importantes dans la conception et la mise en œuvre des machines et l'organisation industrielle. Cependant, l'évolution ou la migration des installations existantes semble difficile à réaliser et les sites de production présentent généralement une très forte hétérogénéité matérielle à tous les niveaux : machines, composants, logiciels et réseaux de communication.

De même, la composition des équipes techniques des bureaux d'études, de méthode et de maintenance n'a pas vraiment évolué et les compétences des acteurs de la conception, de l'exploitation et du maintien en conditions opérationnelles des outils de production est en retard par rapport à la disponibilité sur le marché de ces nouvelles technologies.

Pourtant, dès 2011, le rapport [Kagermann 2013] soulignait l'importance de la formation et de l'accompagnement des acteurs de la chaîne de production industrielle en citant :

> Industrie 4.0 will radically transform workers' job and competence profiles. It will therefore be necessary to implement appropriate training strategies and to organise work in a way that fosters learning, enabling lifelong learning and workplace-based [continuing professional development]. [...] It is likely that Industrie 4.0 will significantly transform job and skills profiles as a result of two trends. Firstly, traditional manufacturing processes characterised by a very clear division of labour will now be embedded in a new organisational and operational structure where they will be supplemented by decision-taking, coordination, control and support service functions. Secondly, it will be necessary to organise and coordinate the interactions between virtual and real machines, plant control systems and production management systems.

Dans ce contexte de pression très forte sur les organisations pour intégrer des évolutions technologiques dans des usines présentant une très grande disparité matérielle et des équipes aux compétences en cours de mutation, nous avons retenu comme contrainte de développement l'usage de composants de contrôle-commande disponibles sur tous les équipements actuels mais qui disposent de puissance de calcul suffisantes pour intégrer des algorithmes de commande issus de la recherche en robotique.

Les sections suivantes présentent les possibilités et les limites du matériel et du modèle logiciel associé à partir desquels nous proposerons notre contribution au développement de systèmes de contrôle-commande.

2.1.2 Des architectures matérielles hétérogènes distribuées

Les lignes de production et les machines industrielles sont des architectures distribuées selon plusieurs aspects. Elles présentent :

- une distribution spatiale des composants électroniques qui vise à rapprocher les blocs d'entrées-sorties des capteurs ou des préactionneurs pour minimiser la longueur de câblage, les risques de défaillance et les perturbations électromagnétiques
- une décomposition fonctionnelle hiérarchique depuis des fonctions « bas niveau » qui traitent les informations issues des capteurs jusqu'à des fonctions de traitement d'information au niveau d'un site de production,

— une distribution fonctionnelle entre de nombreux calculateurs et processeurs, avec, parfois des stratégies de redondance,
— une hétérogénéité entre composants de différents fournisseurs qui échangent des informations à travers plusieurs protocoles de communication.

La figure 2.1 présente deux machines dont les architectures mécaniques sont différentes d'une machine mais qui réalisent la même fonction : ranger des sacs ou des cartons en couches empilées sur une palette. L'une d'entre elles réalise la fonction de palettisation avec une partie opérative spécifique à cette tâche, la seconde exploite un robot industriel qui prend en charge la fonction technique de manipulation et de placement de chaque sac. Ces machines sont développées de façon autonome, mais sont destinées à être intégrées dans une ligne de production. Leurs parties commandes devront interagir avec celles des machines amont et aval (dans le sens du flux de production) et avec un contrôleur industriel en charge de la gestion de la ligne.

FIGURE 2.1 – Exemples de cellules de production. Gauche : palettiseur à couches CETEC, détail : intégration dans une ligne de production. Droite : palettiseur SIDEL intégrant un bras de robot sériel.(©CETEC, ©NEWTEC BAG, ©Sidel).

Le logiciel de contrôle-commande d'une installation automatisée est réparti sur un nombre important de calculateurs et son développement est réparti entre les programmeurs de plusieurs entreprises qui prennent en charge la réalisation :

— de fonctions bas niveau intégrées dans les composants matériels,
— de fonctions de contrôle ou de mise à disposition d'information dans les sous-systèmes,
— de coordination et de gestion des modes de fonctionnement au niveau de l'intégration des sous-systèmes dans une machine ou une ligne de production,
— d'interfaçage avec les logiciels de gestion de production et de gestion d'usine.

Cette situation correspond à la décomposition traditionnelle de la pyramide de l'automation, un modèle de séparation et de description des logiciels présents dans un système de production défini par l' International Society of Automation (USA) en 1995 et porté dans la norme IEC62264-1 en . Ce modèle, illustré sur la figure 2.2 fait apparaître plusieurs niveaux qui affinent la distinction entre les couches opérationnelle (*OT* ou *Operational Technology*) et informationnelle (*IT*, *Information Technology*) :

— Niveau 0 : *Processus de production.* Ce niveau regroupe les informations échangées à haute fréquence avec la partie opérative : données provenant des capteurs et ordres adressés aux

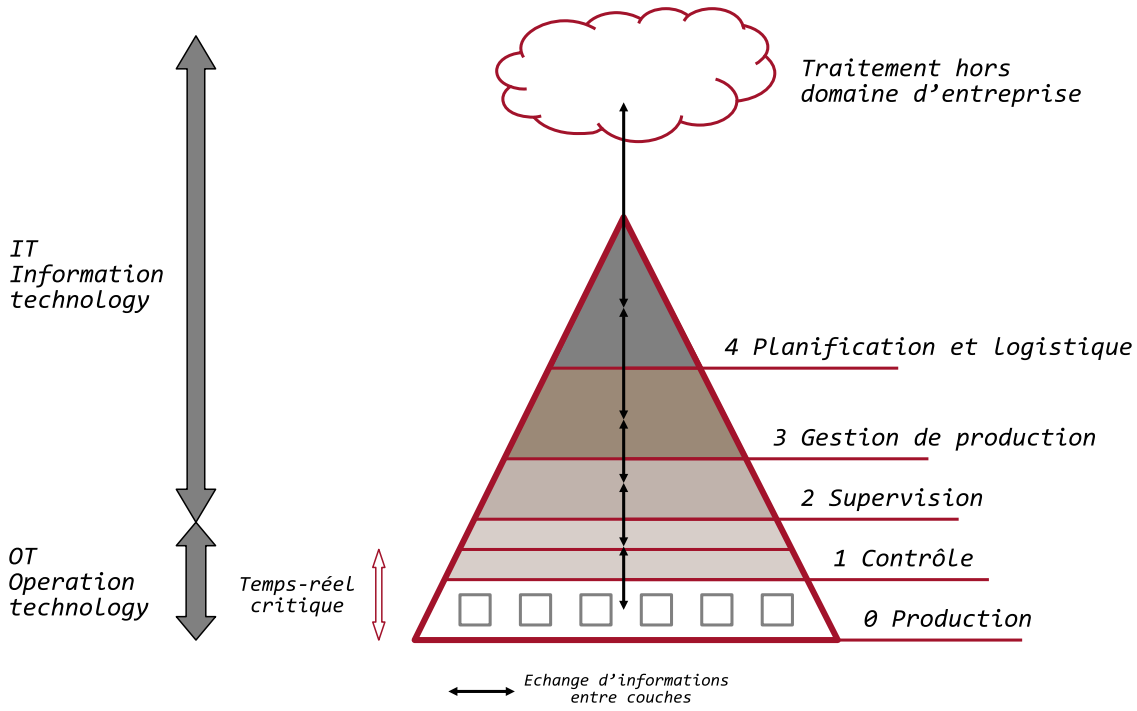


FIGURE 2.2 – Pyramide de l'automation, d'après [Foundation 2021] et [Katti 2020].

préactionneurs. Les traitements logiciels sont embarqués dans les cartes d'entrées-sorties ou les processeurs des contrôleurs de mouvement ou des systèmes de régulation.

— Niveau 1 : *Détection et traitement*. Les actionneurs et les capteurs du niveau 0 sont connectés à des automates programmables. Ces dispositifs exécutent périodiquement, en continu, des programmes qui traduisent le fonctionnement séquentiel attendu des équipements.

— Niveau 2 *Contrôle et supervision*. Le système SCADA, pour *système de contrôle et d'acquisition de données* se trouve à ce niveau.

— Niveau 3 *Opérations de fabrication et gestion*. À ce niveau, les fonctions de gestion sont exécutées par un service de planification. Les logiciels MES (Manufacturing Execution System) sont utilisés pour générer des ordres de production, qui permettent aux opérateurs des services de production, de logistique et de qualité de savoir ce qu'il faut produire et quand il faut le faire.

— Niveau 4 *Planification des activités et logistique*. Les applications logicielles de cet étage sont la planification des ressources de l'entreprise (ERP) et la gestion du cycle de vie des produits (PLM). Les informations générées et disponibles à ce niveau sont utilisées par d'autres services de l'entreprise que le service production : ressources humaines, services financiers, contrôle de gestion, achats, maintenance, par exemple. Dans le modèle traditionnel, c'est également la zone d'échange possible avec des systèmes d'information externalisés.

Ces différents niveaux sont associés à des réseaux de communication différents. Les réseaux

de la zone *operation technology* ont des exigences de robustesse aux perturbations extérieures, de garantie des temps d'échange d'information et de disponibilité de service élevée, les réseaux IT doivent s'intégrer, de façon sécurisée dans le réseau informatique de l'entreprise, fournir des bandes passantes élevées, mais avec un cadre temporel moins contraint.

La décomposition logicielle du logiciel de contrôle d'une machine est réalisée selon deux directions :

— une répartition entre composants : machines, sous-ensembles, actionneurs ou capteurs à capacité de traitement embarquée, passerelles de communication,
— une séparation par niveaux d'abstraction qui part de la couche la plus proche du matériel pour rejoindre la gestion machine et la liaison avec l'étage de supervision et de gestion de production.

Cette fragmentation du logiciel est également associée au partage de responsabilités entre les entreprises qui fournissent les sous-ensembles, aux intégrateurs et aux clients finaux qui prennent en charge le suivi logiciel en phase de production, avec un niveau d'assistance demandée aux sous-traitants qui varie en fonction des ressources disponibles chez l'exploitant.

Sur le plan de la formation et de la culture des différents acteurs, il est possible d'effectuer une distinction entre les automaticiens qui interviennent sur le niveau machine et le service d'informatique industrielle qui prend en charge les développements logiciels à partir des composants de supervision. Cependant, avec l'évolution des pratiques et des outils accompagnant la mise en place de l'approche Industry 4.0, la frontière entre les deux approches est de plus en plus floue. Certains systèmes de contrôle-commande peuvent en effet intégrer des traitements de type « intelligence artificielle » déportés dans le cloud. Inversement, les systèmes de supervision peuvent, via les nouveaux protocoles de communication, interagir directement avec les composants matériels.

Face à la variété de composants disponibles sur le marché, le secteur de l'automation industrielle a cherché à mettre en place une possibilité d'interopérabilité, c'est-à-dire de pouvoir faire fonctionner des composants matériels de fournisseurs différents avec le minimum de développements spécifiques pour l'opérateur et sans perte de fonctionnalités. Dans le cadre de notre étude, cette recherche d'interopérabilité passe par l'usage de réseaux et de protocoles de communication standards.

2.1.2.1 Réseaux industriels temps-réel

Dans le domaine de la commande de processus à dynamique élevée, l'extension du comportement déterministe des logiciels exécutés sur des cibles temps-réel est obtenu par l'usage de protocoles de communication industriels temps-réels. Pour avoir une bande passante élevée, les protocoles modernes sont à support Ethernet. Certains protocoles propriétaires sont utilisés pour la communication entre composants d'un même fournisseur (les réseaux X2X pour B&R Automation ou Mechatronics pour Yaskawa), mais l'interopérabilité recherchée par les fabricants de machine passe par l'usage de l'une des technologies majeures du marché, EtherCAT, POWERLINK, SERCOS III ou PROFINET IRT. Ces solutions concurrentes utilisent des techniques différentes pour surveiller et garantir la périodicité des échanges entre les composants d'un réseau de communication. Elles ont des exigences différentes sur la topologie matérielle du réseau, des niveaux de performances différents (en termes de robustesse aux perturbations et de période minimale d'échange) et des gammes variables de composants qui les mettent en œuvre,

mais tous sont adaptés à la gamme d'applications couvertes par notre étude.

Pour les développements en cours dans l'équipe RoBioSS, le réseau de communication temps-réel utilisé est POWERLINK, développé initialement par B&R Automation puis par l'association EPSG (*Ethernet Powerlink Standardisation Group*). C'est un standard open source basé sur le protocole Ethernet classique, il n'est pas conditionné à l'usage de cartes de communication spécifiques [EPSG 2016]. Un PLC a le rôle de chef d'orchestre (ou *Managing Node*, dans la terminologie du standard) et organise un échange périodique d'informations avec les *Controlled Nodes*, les autres composants présents sur le réseau qui, dans un cycle d'échange, sont interrogés successivement et envoient une réponse accessible à tous les nœuds. Dans la bande passante complémentaire aux échanges périodiques, il est possible d'échanger des informations non prioritaires par une communication Ethernet classique. Le protocole POWERLINK, comme les autres protocoles industriels identifiés précédemment, permet également d'échanger des informations de sécurité entre composants de la chaîne de sécurité.

2.1.2.2 Vers un mécanisme d'échange d'informations universel ?

Dans le cadre des exigences de la mise en œuvre du concept d'Industrie 4.0, l'adoption d'un cadre d'échange d'informations entre tous les composants de la chaîne de production est impératif. Le standard retenu pour cela est OPC-UA, un ensemble de technologies dont le développement est organisé par l'association internationale OPC Foundation. Cette solution permet l'évolution du modèle de la pyramide de l'automation vers le concept d'un réseau industriel étendu, tel que présenté par OPC Foundation et illustré sur la figure 2.3.

Ce concept soutient une décomposition fonctionnelle non hiérarchisée, pour la collecte et le traitement de l'information, mais pas spécifiquement pour la partie contrôle-commande qui nécessite toujours un partage de tâches et de responsabilités clairs. Cependant, cette exigence d'accès de tous les composants matériels les place tous dans la zone d'action de l'Information Technology. Ceci rend possible l'accès direct aux informations utiles à une fonction donnée mais impose des contraintes matérielles et organisationnelles fortes en termes de cybersécurité et de partage de bande passante entre des communications dont la criticité et la période d'échange sont très variables.

Le niveau d'adoption d'OPC UA par les acteurs de l'automatisme et de l'information industrielle en fait un composant indispensable dans le cadre du développement d'un logiciel de contrôle-commande. Pour l'instant, la possibilité d'échanger des informations non temps-réel sont implémentées, ou disponibles dans de nombreux composants. L'ajout de fonctionnalités de surveillance des temps d'échanges d'information est en cours de standardisation, dans le cadre des travaux du groupe *Field Exchange* de la Fondation OPC. L'objectif est d'ajouter les fonctionnalités de la technologie TSN (*Time Sensitive Network*) dans la couche de réseau du modèle de communication pour atteindre des exigences de temps-réel dur à une période garantie d'échange d'informations entre composants de $10ms$.

En complément de l'intérêt d'OPC UA pour garantir l'interopérabilité des composants, cette technologie apporte également des fonctionnalités de :

— sécurité, avec l'intégration de techniques de chiffrement et d'authentification,
— extensibilité, c'est-à-dire que de nouvelles versions d'OPC-UA peuvent ajouter de nouvelles fonctions sans affecter les applications existantes. Il est également prévu de pouvoir ajouter de nouveaux protocoles de communication ou de chiffrement,

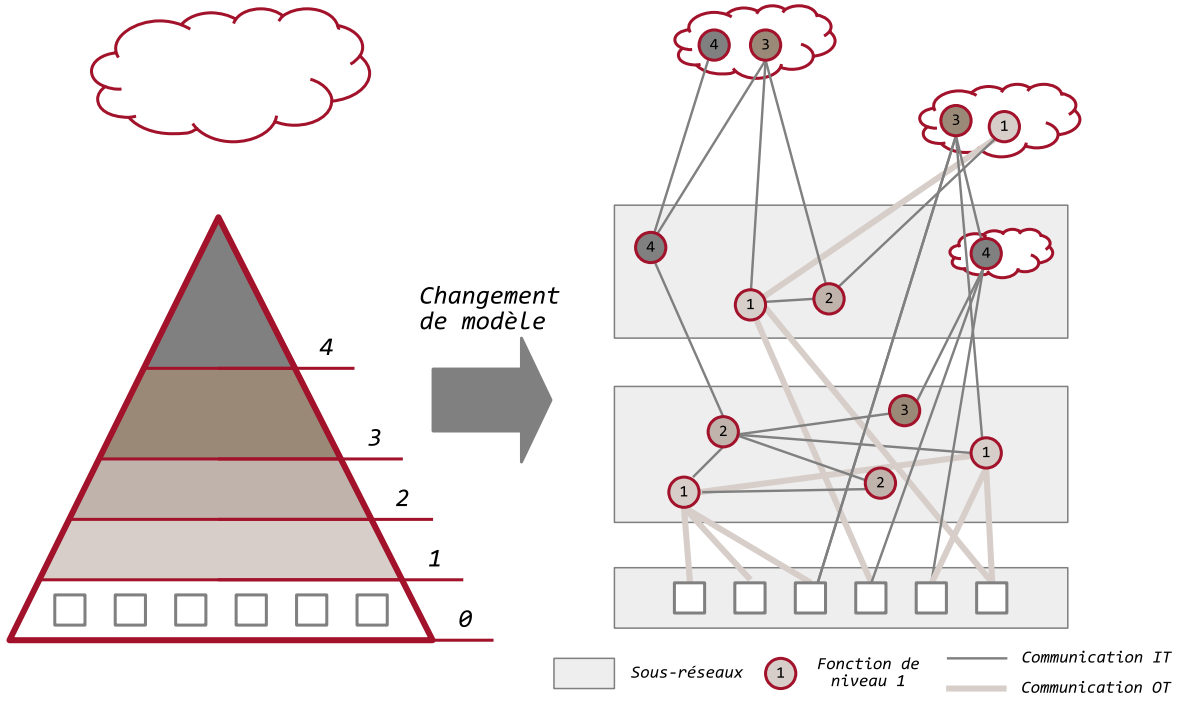


FIGURE 2.3 – Le remplacement de la pyramide de l'automation par un réseau industriel universel, d'après [Foundation 2021].

— choix du mode de communication (client / serveur ou publish / subscribe),
— modélisation des informations et des processus. OPC UA permet d'ajouter des métadonnées aux données échangées et de modéliser les processus pilotés. Le standard offre également la possibilité aux logiciels clients d'explorer et de découvrir les données accessibles fournies par le serveur.

Ce dernier point s'accompagne de la possibilité pour des secteurs d'activités ou des groupes de fournisseurs de développer des exigences complémentaires, appelées *Companion Specifications*. Parmi celles-ci, on peut noter *OPC UA Information Model for IEC 61131-3* rédigée avec PLCopen pour définir la communication entre PLC et *OPC UA for Robotics* , document établi avec l'association VDMA (l'association allemande qui regroupe les entreprises du secteur de la mécanique) qui vise à modéliser les fonctionnalités d'un robot industriel en vue de pouvoir l'intégrer dans un système de production quel que soit le fabricant.

2.1.3 Systèmes industriels temps-réel selon la norme IEC61131

Le développement de nos systèmes mécatroniques se place donc dans le cadre de l'usage des composants standards du secteur de l'automation industrielle. Les calculateurs utilisés répondent aux exigences de la norme IEC61131 qui définit les contraintes auxquelles ces composants doivent répondre et définissent un cadre logiciel qui permet de simplifier le développement des systèmes de contrôle-commande temps-réel basés sur ces architectures.

2.1.3.1 Exigences matérielles

Les calculateurs industriels pour le secteur de l'automation ont suivi les évolutions de l'industrie des composants électroniques tout en conservant les exigences propres à ces composants définies il y a plus de trente ans.

Un automate programmable industriel moderne présente les caractéristiques suivantes :

— une puissance de traitement élevée : les systèmes PLC modernes sont dotés de processeurs identiques à ceux des composants électroniques d'usage général,
— l'aptitude à contrôler un nombre important d'entrées-sorties via une grande variété de bus de communication,
— une conception modulaire qui permet à la fois la personnalisation des équipements et leur évolution au cours de la vie du produit,
— l'aptitude à la mise à l'échelle des installations, ce qui permet de mutualiser une grande partie du développement pour des gammes de machines différentes,
— un cadre de développement logiciel indépendant du fournisseur de matériel avec la mise à disposition impérative de plusieurs langages de programmation et la possibilité d'en proposer d'autres,
— une robustesse et une fiabilité des composants élevées, le matériel est conçu pour résister à des environnements industriels difficiles caractérisés par des perturbations électromagnétiques, des vibrations, de la poussière, des variations de température importantes,
— la mise à disposition d'outils de diagnostics intégrés, pour le développement logiciel, la mise en service et le maintien en condition opérationnelle des machines.

Les spécifications définies par la norme IEC61131 permettent des réalisations très différentes. L'exigence de comportement déterministe des calculateurs se traduit par des solutions concurrentes basées sur des versions adaptées de *différents systèmes d'exploitation* :

— des solutions propriétaires, comme *Siemens S7 Operating System*, pour les PLCs Siemens,
— *VxWorks*, un système temps-réel utilisé dans les applications critiques (présent sur les PLC B&R Automation),
— *Linux*, avec des extensions temps-réel. Ce sont des solutions qui permettent le partage de ressource entre des tâches temps-réel avec du développement logiciel généraliste (OS présents sur le matériel Keba et Bosch Rexroth, par exemple),
— *Windows*, avec des mécanismes préemptifs qui permettent aux tâches de contrôle d'avoir un comportement temps-réel (une partie des contrôleurs Beckhoff)
— *FreeBSB*, une solution alternative pour certains contrôleurs entrée de gamme Beckhoff.

Les nouveaux composants matériels permettent également d'exploiter des mécanismes d'hyperviseur et de faire cohabiter un système d'exploitation généraliste (ou GPOS, *General purpose Operating System*), qui va prendre en charge la gestion de l'IHM ou des fonctions IT, avec un runtime temps-réel (RTOS, *Real Time Operating System*) qui exécute les tâches de contrôle-commande.

Les exigences de fonctionnement des automates programmables définies dans la norme fournissent au développeur d'applications un système d'exploitation qui garantit une application déterministe des tâches et offre la possibilité d'avoir une exécution parallèle des tâches sans confier au développeur la charge de mettre en œuvre le développement de tâches concurrentes.

La figure 2.4 illustre l'exécution des tâches vue par le développeur. Il dispose de classes de tâches dans lesquelles il peut affecter les tâches résultant de la décomposition fonctionnelle du

logiciel. Ces classes de tâches sont associées à un niveau de priorité d'exécution et à une période d'exécution réglable. Le système d'exploitation fournit un ordonnanceur qui alloue le travail du processeur aux différentes tâches en fonction de leur priorité et contrôle la durée d'exécution des tâches pour garantir les périodes d'exécution exigées. En cas de charge de travail excessive, l'exécution des tâches utilisateurs s'arrête et le système redémarre. La phase d'exécution périodique est précédée d'une phase d'initialisations successives des programmes réalisée au démarrage de l'automate qui n'est pas contrainte temporellement.

L'environnement d'exécution fournit également un mécanisme unique de partage d'information entre tâches et entre les cartes d'entrées-sorties et les tâches utilisateurs. Le développeur est responsable du choix des périodes d'exécution des classes de tâches et de l'affectation des programmes aux classes de tâches en relation avec la dynamique des évènements auxquels le système automatisé doit répondre et avec la puissance de traitement du composant matériel qu'il a choisi. En échange, il bénéficie d'un cadre de développement d'applications concurrentes très simplifié.

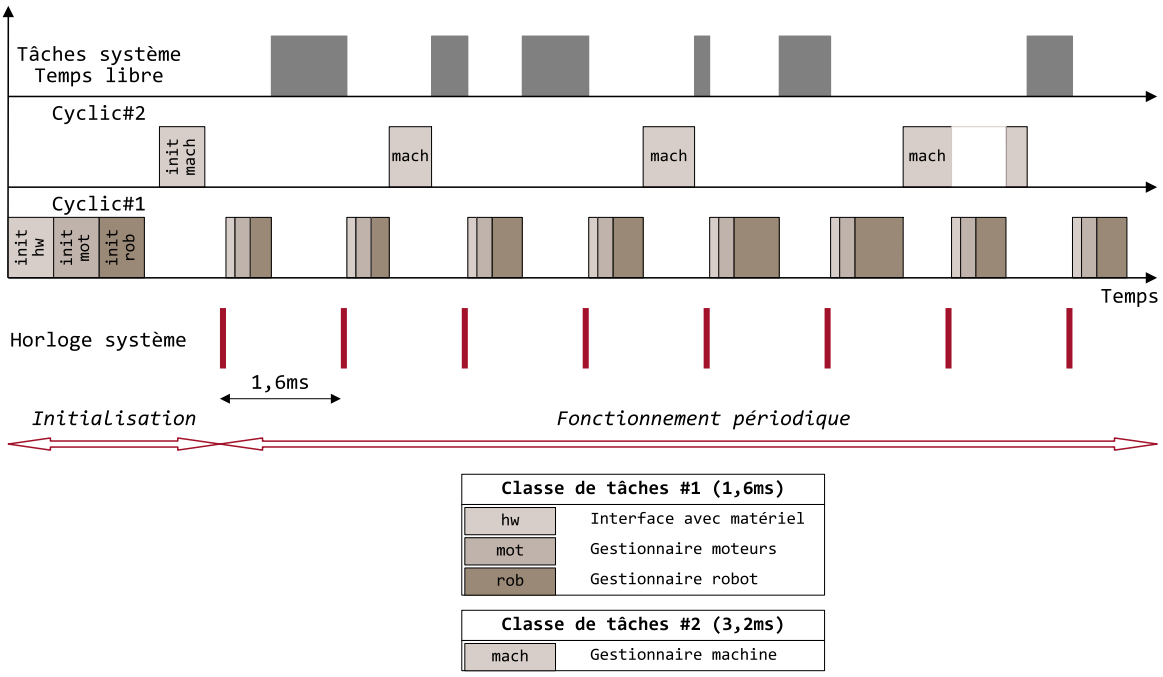


FIGURE 2.4 – Fonctionnement multitâche d'un PLC : répartition de la charge du processeur.

2.1.3.2 Cadre du modèle logiciel

Le modèle logiciel des automates programmables industriels est défini dans la partie IEC61131-3, *Automates programmables - Partie 3 : Langages de programmation*. L'élément de plus haut niveau est appelé *configuration*. Une configuration décrit la structure matérielle d'un système de contrôle commande qui contient des cartes d'entrées sorties et des ressources de traitement (automates ou PC industriels) qui sont capables d'exécuter des programmes.

Ces ressources sont utilisées par des tâches, qui invoquent l'exécution de composants logiciels. Ces derniers sont appelés Program Organization Units (POU) ou unités d'organisation de programme. La norme IEC61131-3 en définit trois types : les programmes, les blocs fonctionnels et les fonctions.

Les programmes sont les unités d'organisation de programme de plus haut niveau, ils peuvent être écrits dans n'importe lequel des langages de programmation de la norme. Chaque programme peut utiliser des fonctions et exécuter des instances de blocs fonctionnels. Les programmes sont exécutés par les tâches, ils peuvent accéder aux variables d'entrées-sorties, à des variables globales et ont leur propre espace mémoire.

Le type de POU le plus courant est le bloc fonctionnel. Il s'agit de segments de code réutilisables qui possèdent un espace mémoire qui leur est associé et qui contient des variables de portée différentes (variables d'entrée, de sortie, d'entrée-sortie et variables internes).

Les fonctions, dans le modèle logiciel des automates, sont des sous-programmes qui n'ont pas d'espace mémoire associé et qui ne peuvent retourner que l'un des types de base (booléen, entier ou nombre à virgule flottante).

La figure 2.5 présente l'un des blocs fonctionnels défini par le standard. Il s'agit du bloc temporisation retardée, `TON` (Timer on Delay). Il possède deux variables de type entrée et deux variables de type sortie. Il doit être appelé à chaque exécution du programme dans lequel il est utilisé. Ses variables d'entrée sont de deux types :

— `PT` contient l'information du délai de retard, c'est une entrée de type *paramètre* ou *asynchrone* : elle doit être définie au moins une fois ou peut être changée occasionnellement,

— `IN` contient l'information booléenne qui déclenche la temporisation à son activation et qui doit rester vraie jusqu'au déclenchement de la sortie. C'est une variable *synchrone* qui doit être actualisée à tous les cycles, avant l'appel du bloc fonctionnel.

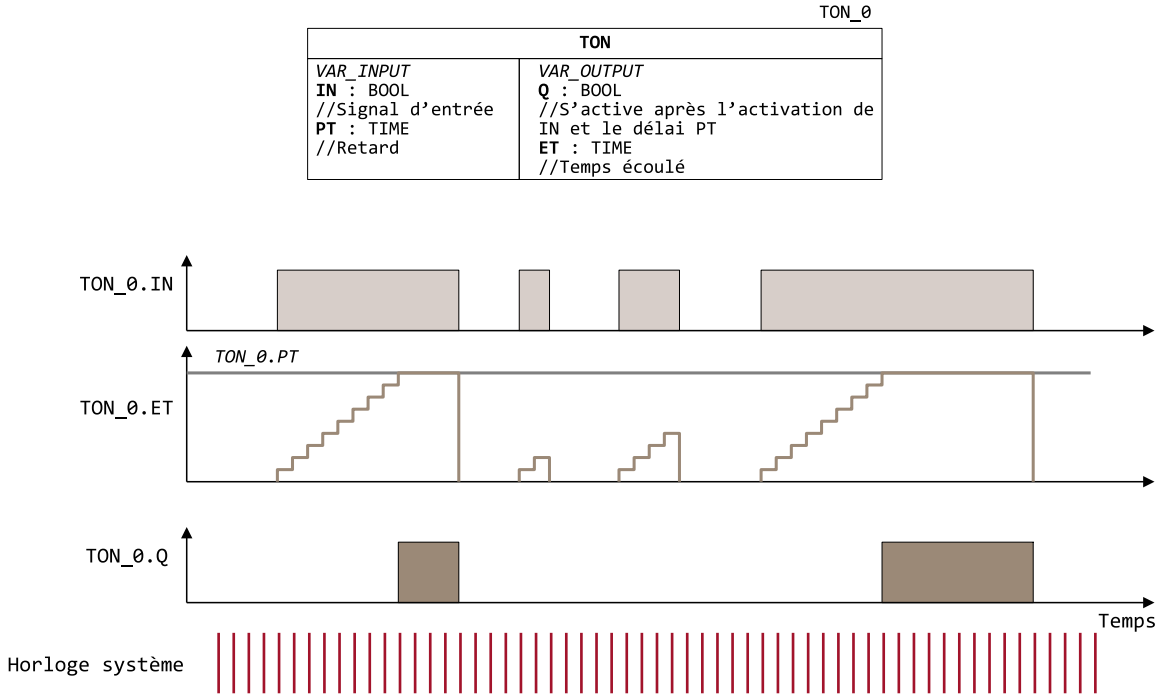


FIGURE 2.5 – Interface et chronogrammes de fonctionnement du bloc fonctionnel standard `TON`.

2.1.3.3 Démarche de structuration du développement logiciel

Comme présenté précédemment, dans le modèle logiciel de la norme IEC 61131, le principal outil d'organisation logicielle est le bloc fonctionnel. En 2017, PLCopen a publié un guide méthodologique pour la création de bibliothèques dans le secteur de l'automation. Ce document est construit à partir d'un retour d'expérience sur les différents standards préalablement publiés par l'association. Ce travail de synthèse a fait apparaître les propriétés nécessaires pour régir le comportement d'un tel composant logiciel dans le contexte d'une exécution périodique.

Les blocs fonctionnels ont vocation à être exécutés à tous les cycles de fonctionnement de la classe de tâches du programme dans lequel ils sont appelés. Ils ont naturellement deux états : un état d'inaction et un état où ils réalisent leur fonction (pour rappel, un bloc fonctionnel est semblable à une classe à une seule fonction). Ce comportement est proche de celui de l'activité associée à une tâche dont un exemple d'organisation du cycle de vie est présenté figure 2.6. Les blocs fonctionnels peuvent être activés ou désactivés. Un troisième état peut être utilement ajouté : il s'agit d'un état d'erreur où le bloc fonctionnel, bien que passé en état actif, ne peut pas réaliser la fonction qui lui est associée. Le programme, ou bloc fonctionnel, qui exécute l'instance de bloc fonctionnel en situation d'erreur peut agir sur la cause de l'erreur, désactiver le bloc fonctionnel et le réactiver.

Un aspect central du document « PLCopen compliant libraries » porte sur la définition d'interfaces unifiées pour l'activation du bloc fonctionnel et le retour d'information sur son état courant. Les premiers standards de l'association PLCopen avaient produit des solutions d'interfaces cohérentes au sein d'un standard mais non unifiées. La figure 2.7 présente deux exemples

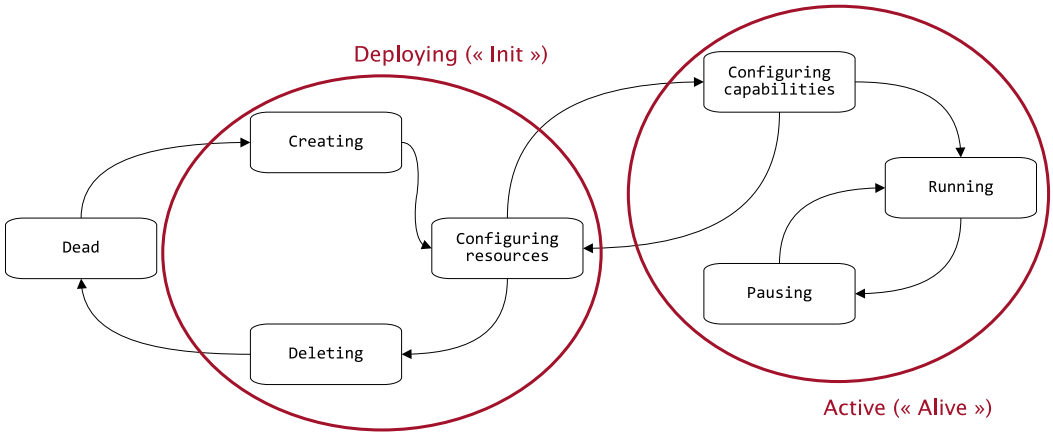


FIGURE 2.6 – Machines d'états du cycle de vie d'une activité [Bruyninckx 2023].

de blocs fonctionnels. Le bloc `MC_MoveAbsolute` est un des blocs fondamentaux du standard PLCopen Motion Control. Il permet de réaliser un mouvement depuis une situation courante `(position, vitesse, accélération)` vers un point d'arrêt défini par l'entrée `Position` avec une loi de commande à vitesse et accélération saturées, définies par les valeurs des entrées `Velocity`, `Acceleration` et `Deceleration`.

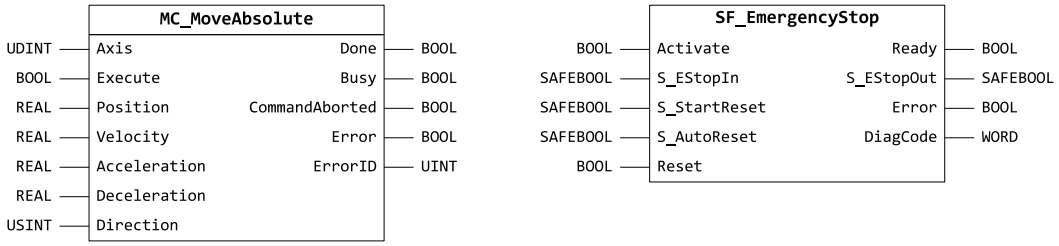


FIGURE 2.7 – Exemple d'interfaces de blocs fonctionnels des standards PLCopen Motion Control [Gotha 2011] et Safety Software [PLCopen 2018].

L'activation de l'entrée `Execute` provoque le démarrage de la fonction qui prend en charge la génération d'une consigne de position. La sortie `Busy` est mise à `True` dès que la fonction commence à être réalisée. Lorsque la destination est atteinte, `Busy` est désactivée et `Done` passe à `True`. Si un autre mouvement est requis, la sortie `CommandAborted` passe à `True`. Si une erreur se produit lors du passage des paramètres ou en cours de mouvement, la sortie `Error` s'active et un identifiant d'erreur est affecté à la sortie `ErrorID`. La remise à faux de l'entrée `Execute` fait sortir la machine d'états du bloc des états correspondants à l'activation des sorties `Done`, `CommandAborted` et `Error`.

Le bloc fonctionnel présenté comme exemple du standard PLCopen Safety est celui qui prend en charge un bouton d'arrêt d'urgence, il s'agit du bloc `SF_EmergencyStop`. L'interface du bloc fait apparaître deux types de variables, les types communs à la programmation d'automatismes,

`BOOL`, par exemple, et des types associés à la réalisation des fonctions de sécurité, `SAFEBOOL` pour cet exemple. Les entrées `Activate` et `Reset` sont présentes dans tous les blocs du standard. L'état de fonctionnement du bloc est décrit par les sorties `Ready` et `DiagCode`.

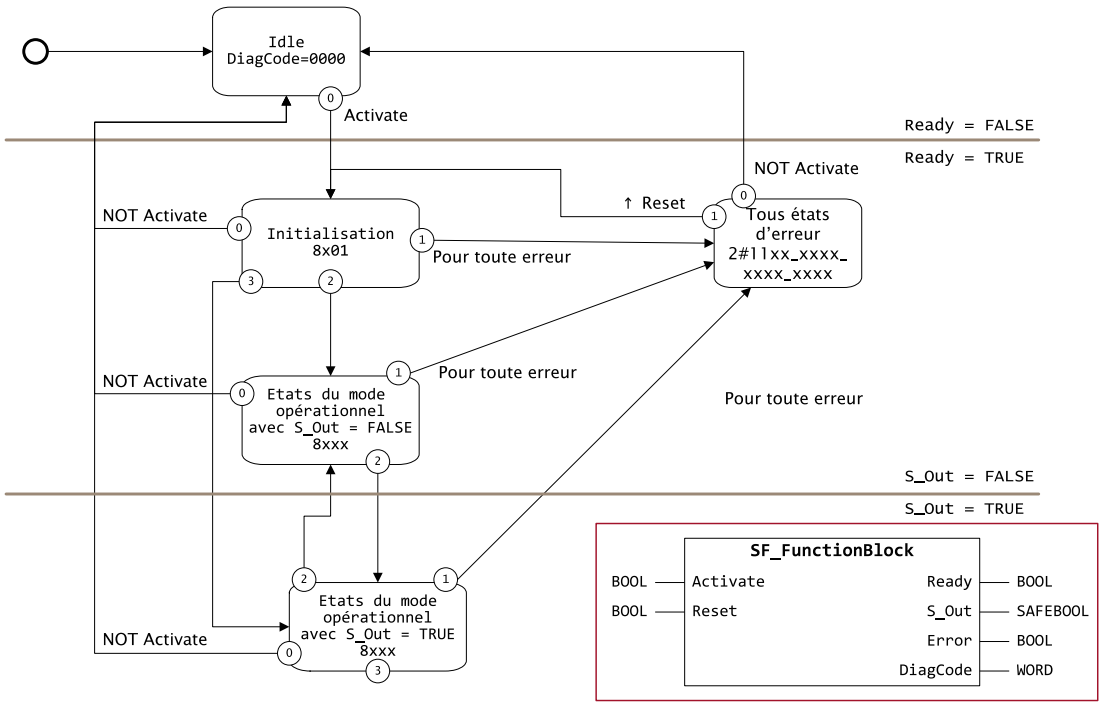


FIGURE 2.8 – Machine d'états générique d'un bloc fonctionnel PLCopen Safety software [PLCopen 2018].

La figure 2.8 décrit la machine d'états générique d'un bloc fonctionnel du standard de logiciels de sécurité. Au démarrage de la tâche qui exécute une instance de ce bloc, celle-ci est placée dans l'état `Idle` et la sortie `Ready` s'active. La mise à vrai de l'entrée `Activate` passe le bloc en situation d'initialisation et `DiagCode = 8000`. Si une erreur apparaît, la machine d'états passe en état d'erreur et l'identité de l'erreur est renseignée dans la sortie `DiagCode`. Dans le cas contraire, le bloc va alterner entre des états opérationnels dont dépendent les valeurs des sorties de sécurité . La sortie `DiagCode` permet d'identifier l'état en cours.

Finalement, l'analyse de ces premiers standards a permis de définir un nombre fini de modèles de comportement de blocs fonctionnels et une solution d'interface unifiée a été choisie dans la proposition du document *PLCopen Compliant Libraries*.

Le comportement d'un bloc fonctionnel destiné à une application d'automatisme appartient à l'une des deux catégories suivantes :

— un actionnement déclenché par l'activation d'une entrée : ce sont les blocs de type `Edge-Trigerred` (référencés par `ETrig` par la suite),

— une autorisation de fonctionnement accordée ou refusée par l'état d'une entrée booléenne pour les blocs de type `Level-Controlled` (`LCon`).

Dans le premier cas, l'entrée qui commande l'activation du bloc fonctionnel et le début de la prise en charge de sa fonctionnalité est nommée `Execute`, dans le second cas, elle s'appelle

Enable. L'interface de sortie présente les différentes informations suivantes :

— Busy : le bloc fonctionnel est actif et réalise la fonctionnalité attendue,
— Done : si un état d'achèvement est présent, le bloc fonctionnel l'a atteint et a terminé son travail,
— Error : le bloc fonctionnel a rencontré une erreur pendant sa phase d'initialisation ou sa phase d'exécution et il ne travaille plus.
— dans ce cas, la sortie ErrorId contient une valeur entière permettant d'identifier l'erreur.

La figure 2.9 présente l'interface et la machine d'état qui régit le comportement d'un bloc de type Edge-triggered.

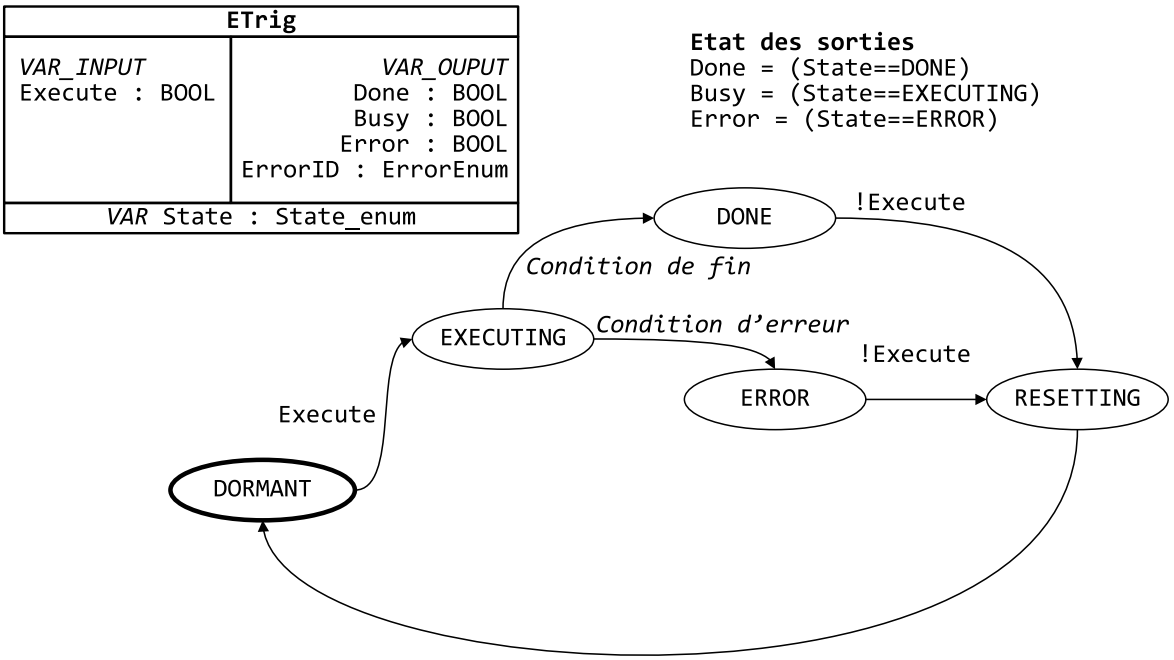


FIGURE 2.9 – Interface et machine d'états générique d'un bloc fonctionnel de type Edge-trigerred.

L'analyse de PLCopen a fait émerger un nombre de types de fonctionnement de base limités, toutes les fonctionnalités à développer dans un logiciel de contrôle-commande peuvent être décrites par l'un de ces modèles. Par rapport aux deux types de base (ETrig et LCon), les variantes portent sur :

— La présence d'une entrée d'interruption : Abort, qui après un état transitoire ABORTING, place le bloc fonctionnel en arrêt dans l'état ABORTED jusqu'à ce que l'entrée Execute soit remise à FALSE.
— La possibilité de limiter la durée de l'état actif avec l'entrée TimeOut. Si le bloc fonctionnel reste dans l'état EXECUTING plus longtemps que cette durée, il passe en ERROR avec une erreur de type TimeOut
— La possibilité de limiter le temps d'exécution de la méthode cyclique du bloc fonctionnel

à la durée définie dans l'entrée `TimeLimit` pour contrôler la charge demandée au processeur : si l'exécution du bloc fonctionnel dure plus longtemps, elle doit s'interrompre pour reprendre au cycle suivant.

La figure 2.10 présente deux exemples de variantes de modèles de fonctionnement. Les éléments en rouge correspondent aux modifications par rapport aux modèles `ETrig` et `LCon` de base. Le premier bloc possède l'entrée `Abort` qui permet d'interrompre son fonctionnement. Le second ajoute les fonctionnalités de limite du temps d'exécution au cours d'un cycle (la condition `Time limit`) et de surveillance du temps de traitement de la fonctionnalité (`Time out`).

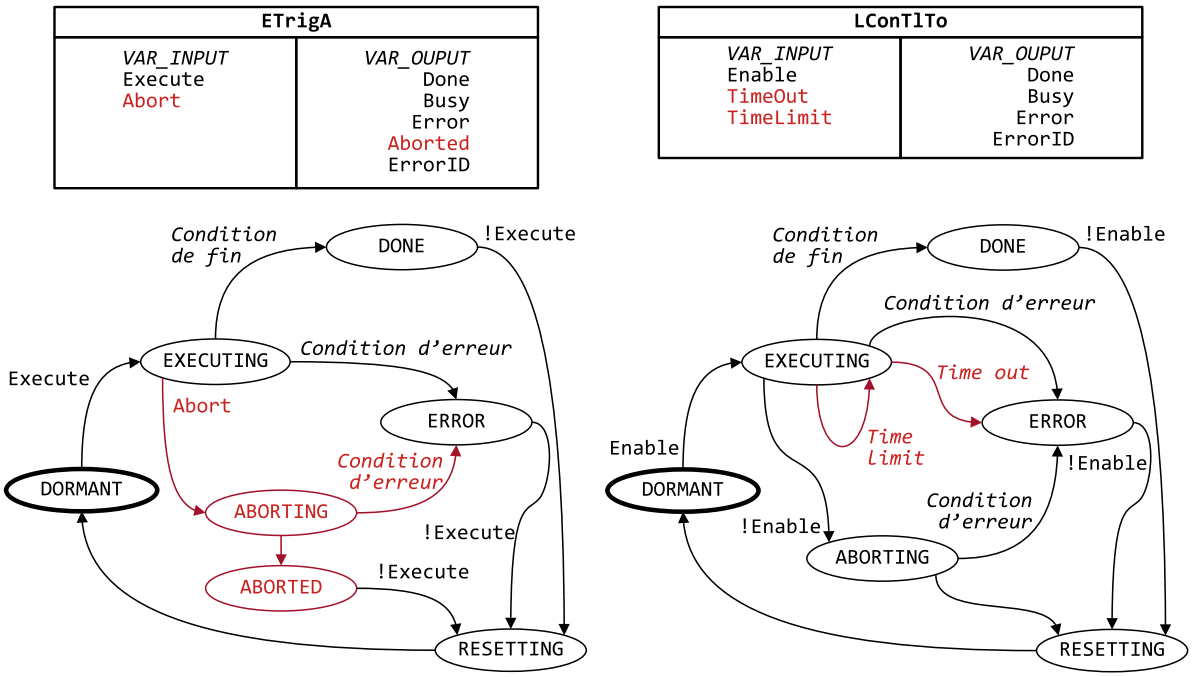


FIGURE 2.10 – Interfaces et machines d'états génériques de blocs fonctionnels de types `Edge-trigerred with abort` et `Level-controlled with time limit and time out`.

2.1.4 Standardisation logicielle de la génération de mouvement

Le développement des logiciels de pilotage d'axes par des contrôleurs de mouvement a fait l'objet d'un travail de standardisation de PLCopen, dès le début des années 2000 (la publication de la première partie du standard date de novembre 2001). Cette approche a permis le développement d'un très grand nombre de solutions d'entraînement électrique avec des technologies différentes mais qui pouvaient être mises en œuvre avec des calculateurs industriels d'autres marques et avec un développement de la partie logicielle générique et facilement modifiable d'une marque à l'autre.

Le terme « contrôleur de mouvement » est utilisé ici pour désigner le *servodrive* où préactionneur intelligent qui réalise l'adaptation de puissance permettant d'alimenter un actionneur électrique en réalisant les boucles d'asservissement courant - vitesse - position.

Le standard PLCopen Motion Control a été adopté par tous les acteurs de l'automation industrielle. Ce document est composé de plusieurs parties :

— Partie 1 : blocs fonctionnels pour le *motion control*. Cette partie regroupe les parties 1 et 2 de la première version du standard, la version 2.0 a été publiée en mars 2011.
— Partie 3 : directives utilisateurs, cette partie présente des cas d'usage des blocs fonctionnels de la bibliothèque.
— Partie 4 : mouvement coordonné. Ce document décrit l'approche et les blocs fonctionnels pour le pilotage d'architectures de type robotique, il est actuellement en cours de révision et son contenu est détaillé dans la chapitre 3.1.1 de ce manuscrit.
— Partie 5 : procédures de prises d'origine
— Partie 6 : extensions pour les systèmes hydrauliques.

2.1.4.1 Structure du standard

Le standard PLCopen Motion Control décrit un mode de contrôle des préactionneurs de motorisation électrique indépendant du matériel et impose aux constructeurs de fournir un ensemble de blocs fonctionnels qui donnent accès aux fonctions élémentaires de préparation et de génération des mouvements. Les interfaces et les comportements de ces blocs fonctionnels sont également imposés et ils doivent pouvoir être implémentés dans les langages de la norme IEC61131-3. Les fournisseurs de matériel peuvent ajouter des fonctions propriétaires complémentaires.

Le lien entre un bloc fonctionnel et l'actionneur qui lui est associé est créé par le passage de l'adresse d'une structure dont le type dépend du fournisseur de matériel. La notion d'« axe » utilisée dans ce standard décrit le mouvement d'un degré de liberté d'un mécanisme, en relation avec un moteur par un système de transformation de mouvement à loi proportionnelle. Il ne s'agit donc pas nécessairement du contrôle direct de l'arbre moteur ou du chariot d'un moteur linéaire, mais il peut décrire le mouvement de la sortie d'un réducteur ou un mouvement de translation, si la chaîne de transmission présente une transformation entre un mouvement de rotation et un mouvement de translation. L'environnement de programmation permet d'associer cette variable à un composant matériel, de paramétrer les limites admissibles de l'axe et éventuellement la description de la chaîne de transformation de mouvement. Il offre également au développeur les outils permettant de configurer et de régler les paramètres de régulation du contrôle en position réalisé par le contrôleur de mouvement.

Les blocs fonctionnels disponibles se partagent en deux catégories :

— les blocs fonctionnels d'administration (`MC_Power`, `MC_Home`, `MC_ReadStatus`, ...),
— les blocs de mouvement (`MC_MoveAbsolute`, `MC_Halt`, `MC_TorqueControl`, ...).

Les principales fonctions réalisées sont les suivantes :

— Préparation de l'axe :
  — `MC_Power` : démarrage et arrêt de l'étage de puissance,
  — `MC_Home` : « prise d'origine », mise en référence du capteur de position par rapport à la position physique de l'axe,
— Information : `MC_ReadStatus` : état de la machine d'états PLCopen Motion Control,
— Modification du mouvement :
  — `MC_MoveAbsolute` : déplacement vers une position absolue,
  — `MC_MoveVelocity` : accélération/décélération puis déplacement à vitesse constante,

  - `MC_Halt` : arrêt contrôlé,
  - `MC_TorqueControl` : application d'un couple constant atteint avec une rampe de couple,
- Coordination d'axes :
  - `MC_GearIn` : activation d'un couplage constant (rapport de vitesses imposé) de l'axe contrôlé avec un axe maître,
  - `MC_CameIn` : engagement d'une came électronique entre l'axe contrôlé avec un axe maître. Une relation polynomiale périodique relie alors la position de l'axe suiveur à celle de l'axe maître, comme le réaliserait une came mécanique.

2.1.4.2 Diagramme d'états PLCopen Motion Control

Le diagramme d'états décrit sur la figure 2.11 est une couche d'abstraction de l'état réel de l'axe. Il définit le comportement de l'axe lorsque plusieurs blocs fonctionnels de contrôle des mouvements sont activés « simultanément ». Cette combinaison de profils de mouvement est utile pour construire un profil plus complexe ou pour gérer les exceptions au sein d'un programme. La règle de base est que les commandes de mouvement sont toujours prises de manière séquentielle, même si l'automate a la capacité d'effectuer un véritable traitement parallèle. Ces commandes agissent sur le diagramme d'état de l'axe, qui se trouve toujours dans l'un des états définis, suivant une logique d'abandon de la fonction en cours ou de stockage dans un tampon des nouvelles commandes. Toute commande de mouvement qui provoque une transition change l'état de l'axe et, par conséquent, modifie la façon dont le mouvement actuel est réalisé.

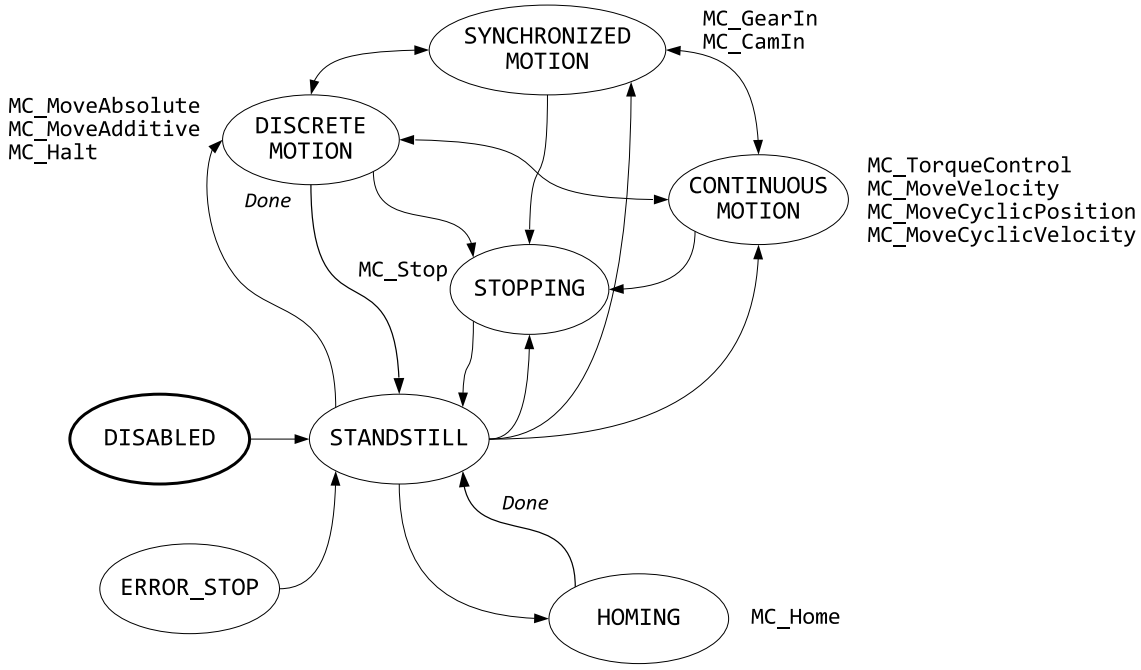


FIGURE 2.11 – Machine d'états d'un axe selon PLCopen Motion Control.

Les différents états sont définis de la façon suivante :
— `Disabled`. Le contrôleur du lecteur est désactivé.
— `Standstill`. Le contrôleur de mouvement n'exécute actuellement pas de mouvement et est prêt à générer des commandes de positionnement.
— `Homing`. Le contrôleur exécute une procédure de prise d'origine.
— `ErrorStop`. Le contrôleur est à l'arrêt après une erreur.
— `Stopping`. L'entraînement arrête le mouvement actif.
— `Discrete Motion`. Le contrôleur exécute un mouvement vers une position cible. Le mouvement a donc une fin définie.
— `Continuous Motion`. Le contrôleur exécute un mouvement sans position cible, le mouvement n'a pas de fin définie.
— `Synchronized Motion`. L'axe est couplé à un autre axe.

2.1.4.3 Blocs fonctionnels de mouvement

Le bloc fonctionnel de mouvement le plus utilisé est le bloc `MC_MoveAbsolute`. S'il est activé dans un état dans lequel son activation est valide, il prend la main sur la réalisation du mouvement interrompt l'exécution d'un éventuel bloc fonctionnel en cours d'utilisation et organise le déplacement de l'axe vers une position d'arrêt définie par une position absolue et des niveaux de vitesse et d'accélération maximale admissibles.

Depuis la version 2.0 du standard *Function blocks for motion control*, les blocs de mouvement ont une entrée complémentaire qui va décrire la situation d'interruption d'une fonction de mouvement par une autre. Avec l'entrée `BufferMode` le bloc fonctionnel peut conserver son fonctionnement standard d'abandon du mouvement en cours et de reprise par un autre à partir de la situation position - vitesse - accélération courante ou temporiser la prise en charge de la seconde fonction de mouvement. Dans ce dernier cas, il est possible de choisir la transition entre les mouvements lorsque le premier est terminé (position ou vitesse objectives atteintes, suivant les cas). L'entrée `BufferMode` peut prendre les valeurs suivantes :
— `mcABORTING` : fonctionnement par défaut : prise en compte immédiate de la nouvelle commande
— `mcBUFFERED` : déclenchement du second mouvement lorsque le premier est terminé
— `mcBLENDING_LOW` ou `mcBLENDING_HIGH` La transition entre mouvements s'effectue à la vitesse minimale / maximale entre les valeurs des deux blocs fonctionnels.
— `mcBLENDING_PREVIOUS` ou `mcBLENDING_NEXT` La transition entre mouvements s'effectue à la valeur de vitesse du premier ou du deuxième bloc fonctionnel.

Lorsqu'un autre mode que le mode `mcABORTING` est choisi, la génération de profil de mouvement ne peut plus être réalisée par le contrôleur de mouvement et est prise en charge par l'automate. Le contrôleur de mouvement passe en suivi de consigne de position cyclique.

Ce principe de fonctionnement qui prévoit la possibilité de mémoriser une nouvelle commande de mouvement et d'en retarder l'action diminue l'aptitude du programme à avoir un comportement déterministe mais sera repris dans la partie du standard portant sur la coordination multiaxe dans l'objectif de générer des trajectoires à partir de mouvements élémentaires (cf. partie 3.1.1.1).

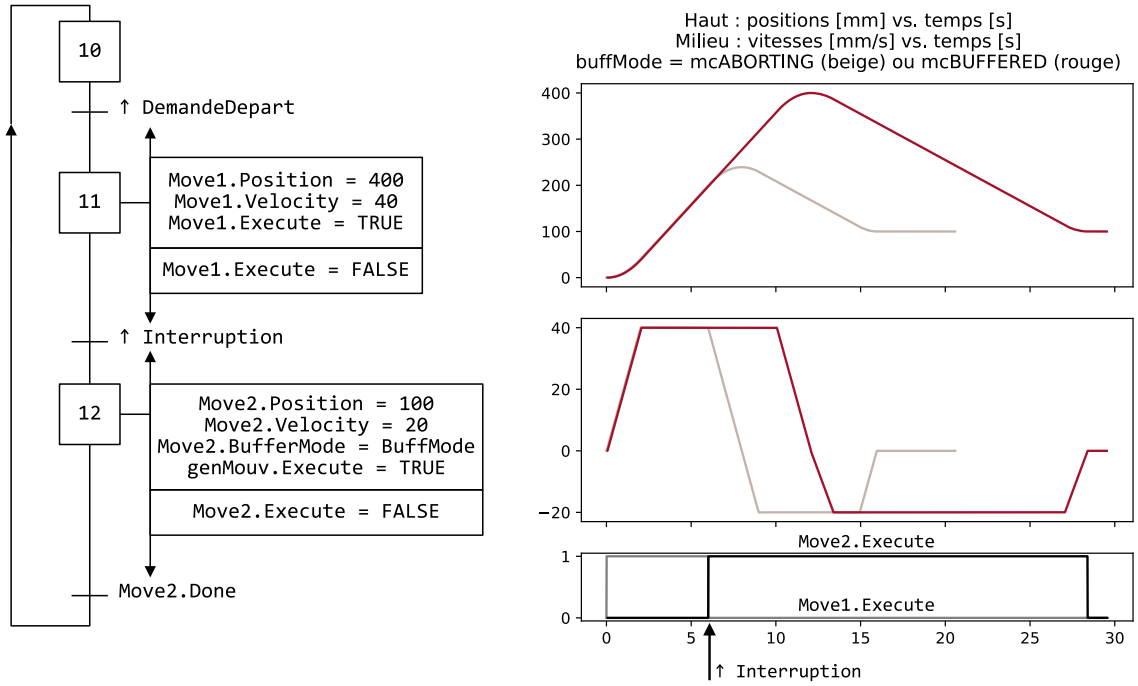


FIGURE 2.12 – Exemple de fonctionnement de l'entrée BufferMode pour l'enchaînement de deux blocs fonctionnels MC_MoveAbsolute.

2.2 rtrmac : un outil de développement de systèmes de contrôle-commande temps-réel

Pour répondre à la demande de la réalisation de logiciels de contrôle-commande pour de nouveaux dispositifs mécatroniques, l'équipe RoBioSS a développé un framework logiciel nommé `rtrmac`, selon l'acronyme de *Real-Time Robotic MultiAxis Control*. Les premières versions ont été réalisées par Philippe Vulliez, dans la décennie 2010 [Vulliez 2019], [Fischer 2019]. Ce framework est plus précisément une bibliothèque logicielle extensible qui comporte les fonctions nécessaires à la réalisation complète d'un système de contrôle-commande multiaxe orienté robotique. Il a fait l'objet d'une démarche de protection industrielle de la part du CNRS sous la forme d'un dépôt à l'agence de protection des programme en juillet 2018 [Vulliez 2018].

Les évolutions de cette bibliothèque ont accompagné le développement des systèmes mécatroniques par l'équipe au cours des vingt dernières années. Le point de départ de la création de cette solution était la décision de mettre en œuvre des composants de contrôle commande industriels standards pour réaliser la partie commande des nouveaux robots ou bancs d'essais. Les bénéfices attendus de ce choix stratégique étaient les possibilités :

— d'exploiter des systèmes temps réels durs avec les fonctionnalités des systèmes d'exploitation de PLC présentés au chapitre 2.1.3,
— d'avoir des composants robustes avec une haute disponibilité,
— de minimiser le besoin de développement de cartes électroniques spécifiques,

— de standardiser, en interne, la méthodologie de développement logiciel et de créer des bibliothèques réutilisables entre projets pour accélérer la conception logicielle et exploiter plus facilement les algorithmes implémentés dans l'équipe.

La première version du framework était rédigée en langage IEC Structured Text, la deuxième en C et la troisième, en cours de développement, en C++. Ces changements sont permis par l'évolution des environnements de programmation automate qui permettent de plus en plus de développer des briques logicielles en C++ en conservant la compatibilité avec les langages pour automates.

Une partie de travaux de recherche s'inscrivent dans l'extension de la bibliothèque `rtrmac` au développement d'interfaces de nouveaux capteurs et au pilotage de groupes d'axes dans l'objectif d'unifier la démarche de conception de la couche *coordination multiaxe* de nos nouveaux dispositifs mécatroniques. Cette approche est détaillée dans le chapitre 3 de ce document.

Avant l'intégration de ces nouvelles fonctionnalités, j'ai mis en œuvre la bibliothèque `rtrmac` dans le cadre du développement de plusieurs dispositifs. A titre d'exemple, il est possible de citer la remise à jour du système de contrôle du système locomoteur du robot humanoïde de grande taille, ORHRO, qui a fait l'objet du travail de thèse de doctorat de Jérémy Gastebois [Gastebois 2017]. Ce robot à quinze degrés de liberté a une armoire de commande comportant un automate programmable industriel et huit contrôleurs de mouvement standard, de marque B&R Automation.

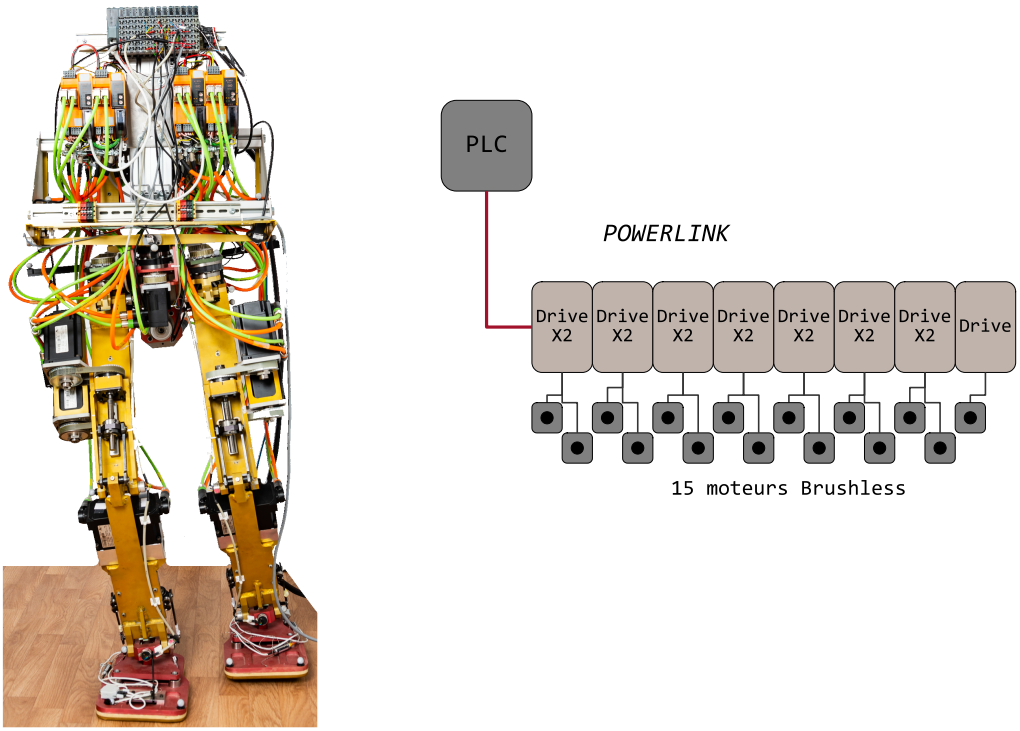


FIGURE 2.13 – Robot humanoïde ORHRO et schéma de l'architecture de commande.

Dans le cadre de ce projet, nous avons exploité la possibilité de mettre à l'échelle la partie commande et le logiciel, comme présenté au chapitre 2.1.2 : le développement d'un algorithme

de stabilisation posturale a été réalisé et implémenté sur une jambe avant d'être intégré dans la commande du système complet, deux jambes et torse.

Un second exemple permet de montrer la versatilité du framework `rtrmac`. Il s'agit du développement de logiciels de contrôle de bancs de test d'arc et de flèches exploités dans le cadre des travaux de thèse de doctorat d'Andrian Kuch (cf. figure 2.14). Pour ces projets, la partie *génération de mouvement* était plus simple à réaliser, mais les architectures retenues permettent de fusionner simplement les mesures issues des contrôleurs de mouvement et des capteurs complémentaires.

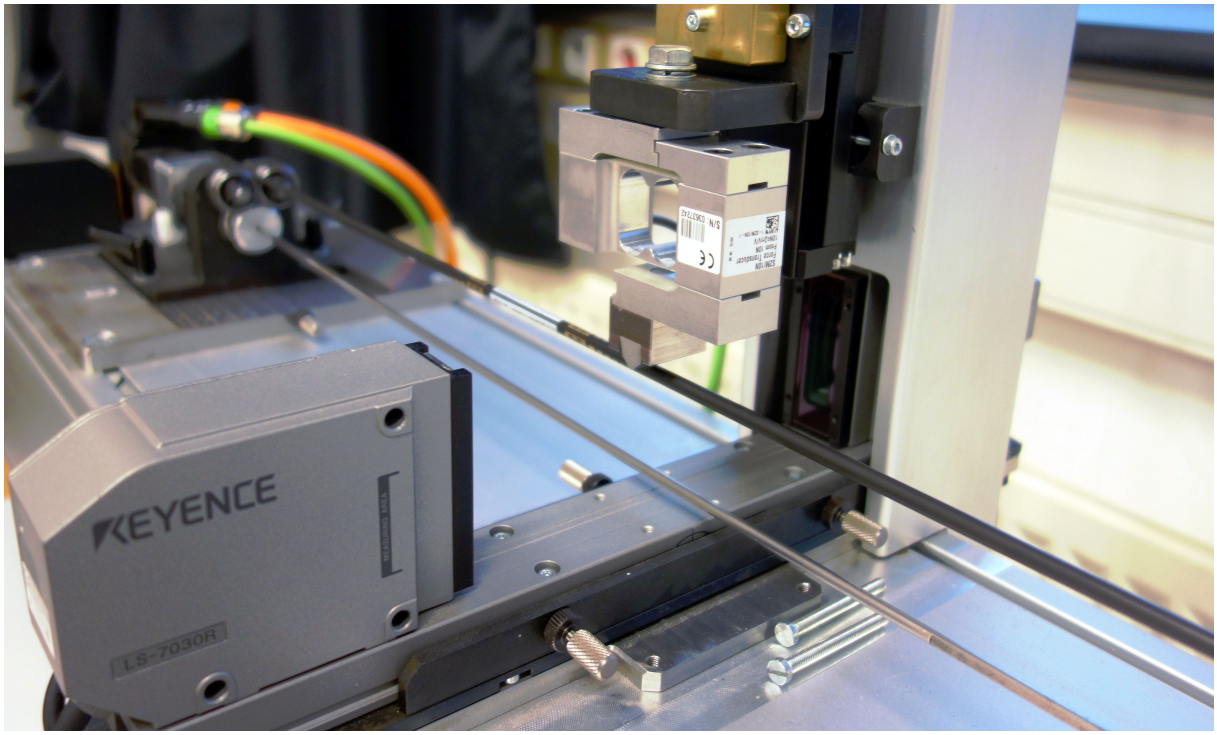


FIGURE 2.14 – Banc de test de flèches, CAIPS (CREPS de Boivre), FFTA.

Les deux principes fondamentaux de cette bibliothèque sont la définition d'un cadre de création de classes adaptées à une exécution périodique sur des cibles temps-réel industrielles et la création d'un axe robotique qui permet d'abstraire un comportement unifié quelle que soit la technologie d'entraînement électrique utilisée.

2.2.1 Structure de la bibliothèque

Le principe central de la construction de la bibliothèque `rtrmac` est de garantir à toutes les classes qu'elle contient un mécanisme permettant de :

— Vérifier la validité des paramètres des fonctions et de contrôler les situations d'actualisation des paramètres.

— Interrompre le fonctionnement des objets en cas d'apparition d'une erreur, d'identifier l'erreur et de transmettre l'information de la situation d'erreur à l'unité logicielle qui

exécute l'objet. Ce mécanisme permet de mettre en œuvre une stratégie de sortie des situations d'erreur.

— Suivre un comportement unifié pendant l'exécution cyclique de la méthode principale.

Cette approche met en application les exigences détaillées dans les paragraphes précédents :

1. Exécution des programmes dans un système d'exploitation temps-réel industriel,
2. Possibilité de pouvoir utiliser les langages de la norme IEC 61131-3 dans le reste du projet logiciel,
3. Respect des recommandations PLCopen pour la réalisation de bibliothèques de blocs fonctionnels.

Le deuxième point de cette liste impose une communication entre programmes par l'intermédiaire de structures d'échange compatibles avec les exigences de la norme de programmation des automates. Ce fonctionnement par flux de données sera utilisé de façon identique au sein d'une même tâche et entre programmes, un exemple sera présenté figure 2.23 lorsque nous présenterons l'implémentation de la régulation des axes électriques.

Les première et troisième exigences sont satisfaites en créant des classes :

— dont la méthode principale est destinée à être exécutée de façon périodique et doit implémenter l'une des machines d'états de la recommandation PLCopen,

— et qui présentent les entrées de déclenchement et d'inhibition et les sorties d'informations génériques.

Dans la suite de ce document, pour désigner ce groupe de classes permettant d'obtenir un fonctionnement similaire à un bloc fonctionnel, on utilisera le terme générique *unité d'organisation logicielle*.

Pour mettre en œuvre ce principe, la création d'une nouvelle unité d'organisation logicielle requière la création de deux classes : la classe qui va réaliser périodiquement la fonctionnalité attendue et une classe d'assistance qui va vérifier la validité des paramètres renseignés. Ces deux classes doivent hériter de deux classes de base abstraites génériques `RbFbAbstractParam<CParam>` qui définit le comportement d'une classe de contrôle-commande et `RbParam<Param_typ>` qui met en place le mécanisme de contrôle d'une structure de paramètres.

La classe `RbFbAbstractParam` est une classe abstraite qui implémente la structure générale d'une tâche de contrôle-commande, elle possède une méthode centrale `Cyclic` qui, comme pour un bloc fonctionnel du modèle IEC 61131 doit être exécutée à tous les cycles automate. Elle a pour membres principaux une instance de la classe de gestion de paramètres génériques `RbParam` et une instance d'une horloge système qui peut contrôler la durée d'exécution de la tâche.

Dans la méthode cyclique, est implémentée une version abstraite des machines d'états recommandées par le document *PLCopen Compliant Librairies* et présentées au chapitre 2.1.3.3. La classe `RbFbAbstractParam` est dérivée une première fois pour définir un comportement de type `Level Controlled` ou `Edge Triggered`. Les différentes options comme la présence d'une entrée d'abandon (`Abort`) ou la surveillance de la durée de l'état `Executing` sont également implémentées. La seule modification par rapport au diagrammes d'états de la recommandation PLCopen est l'ajout d'un état `Enabling` qui permet de passer plusieurs cycles automate dans la phase d'initialisation et de partir dans l'état d'erreur, si nécessaire, sans passer par l'état de travail normal (`Executing`) (cf. figure 2.15).

La figure 2.16 présente l'implémentation du mécanisme de définition des modèles de comportements de blocs fonctionnels. La classe `RbFbAbstractParam` est une classe abstraite et gé-

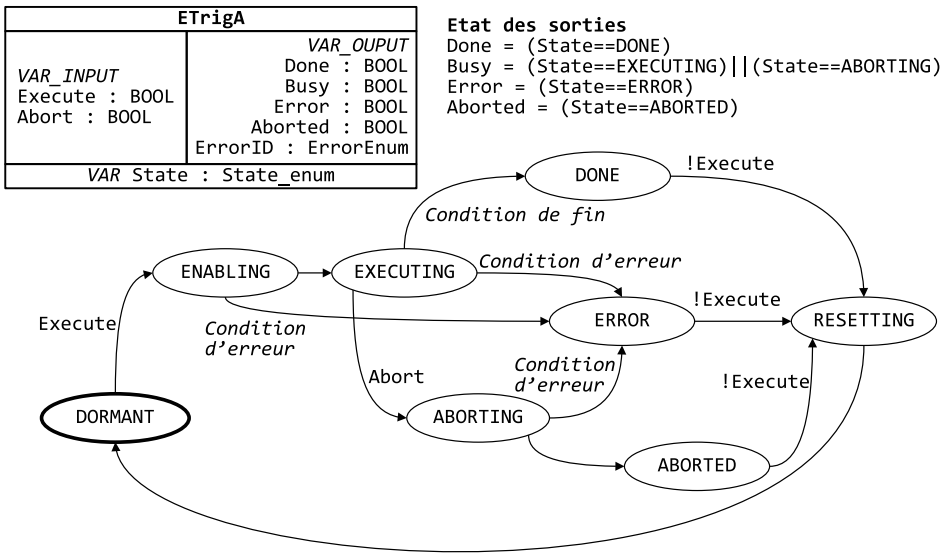


FIGURE 2.15 – Diagramme d'états d'un bloc fonctionnel de type `Edge-triggered with Abort` dans la bibiothèque `rtrmac`.

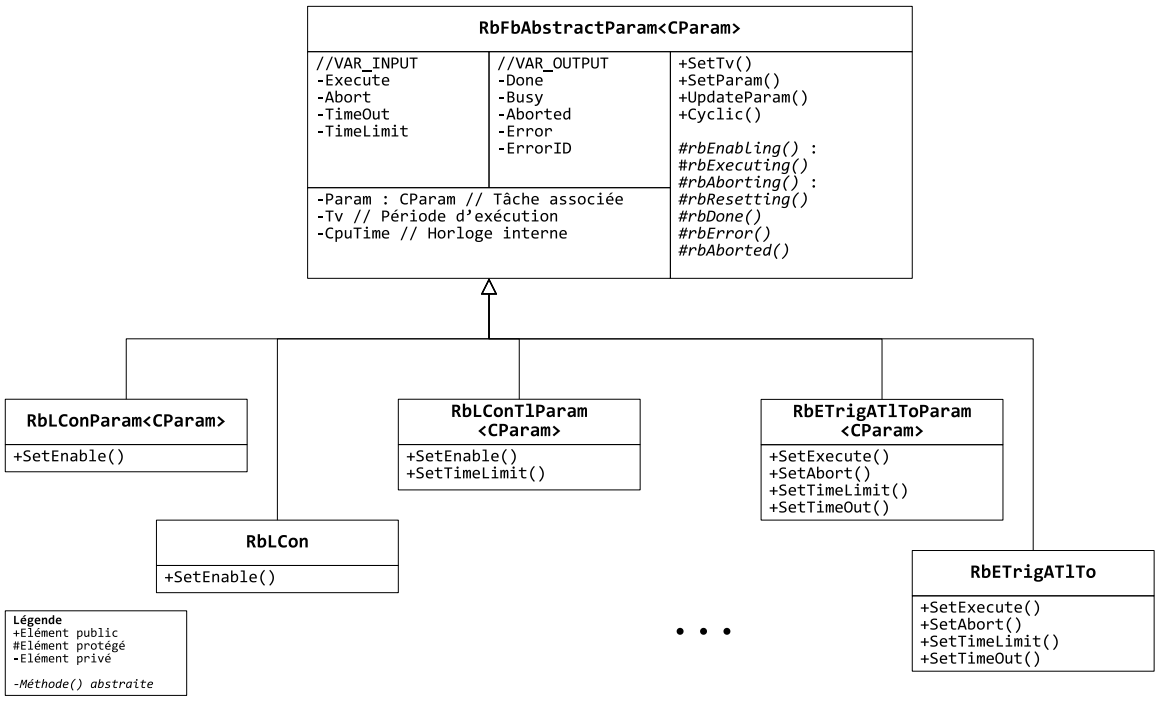


FIGURE 2.16 – Classe de base RbFbAbstractParam et quelques classes dérivées.

nérique : elle dépend du *template* d'une classe d'un type indéfini qui sera particularisé lors de la création d'une classe fille. La classe `RbParam` qui lui est associée est également une classe tem-

plate : elle dépend d'un type variable qui sera particularisé lorsque l'implémentation fournira la structure de paramètre de la classe fille.

La classe `RbFbAbstractParam` implémente la machine d'états et l'interface les plus complètes. Elle est dérivée en classes également abstraites qui codent les différents comportements et interfaces présentés dans le document *PLCopen Compliant Librairies*.

Les données membres de la classe de base sont les entrées-sorties génériques (`Execute`, `Done`, `Error` ...), la classe de vérification de paramètres, la période d'exécution de la classe de tâches dans laquelle la tâche sera exécutée. Elle présente plusieurs méthodes importantes : `Cyclic()` est la méthode principale qui s'exécute périodiquement, les méthodes `SetParam()` et `UpdateParam()` définissent les situations où la structure de paramètres est prise en compte pour la première fois ou mise à jour par la suite. Les méthodes abstraites `rbEnabling`, `rbExecuting()`, `rbAborting()`, `Done()`, `Error()` sont à redéfinir dans les classes dérivées. Elles sont exécutées dans les différents états de la machine d'états et implémentent le comportement des nouvelles unités d'organisation de programme.

Les classes dérivées de premier niveau sont également des classes abstraites. Elles ajoutent aux méthodes de la classe de base les méthodes spécifiques aux entrées-sorties utiles, par exemple `SetAbort()` ou `SetTimeLimit()` qui régissent l'activation de l'entrée `Abort` et l'affectation d'une valeur à l'entrée `TimeLimit` d'une classe du type `rising edge triggered with abort and time limit` (`ETrigATl`). Enfin, ces modèles de classe existent avec ou sans la classe de vérification de paramètres associée (ce sont les variantes `RbClasseX` et `RbClasseXParam`).

2.2.1.1 Contrôle des paramètres

La classe `RbParam<Param_typ>` met en place le mécanisme de contrôle de la structure `Param_typ` contenant les paramètres spécifiques de la nouvelle unité d'organisation logicielle par l'intermédiaire d'une méthode abstraite `Verify`, à particulariser dans les classes dérivées. L'objectif de cette classe est de vérifier l'intégrité des paramètres renseignés et la compatibilité de leurs valeurs par rapport à leur usage dans les algorithmes. En particulier, il faut contrôler que les adresses utilisées dans les passages par référence sont valides et que les valeurs des paramètres ne vont pas provoquer de division par zéro. Ces deux erreurs provoquent en effet une défaillance qui provoque l'arrêt d'un contrôleur industriel. La classe `RbFbAbstract` définit les situations où ces paramètres peuvent être initialement définis ou mis à jour et exécute à nouveau la méthode de vérification. Si celle-ci échoue, l'objet passe dans l'état `ERROR` et l'identifiant de l'erreur est attribué à la sortie `ErrorID`.

2.2.1.2 Exemple

La figure 2.17 présente un exemple de construction d'une unité d'organisation logicielle. Il s'agit d'une brique de base de la bibliothèque *rtrmac* dont la fonction est de convertir la valeur entière retournée par une carte d'entrée analogique en valeur numérique correspondant à l'unité de travail choisie. Par exemple, si la carte d'entrée est reliée à une sonde de température, la sortie `S_ana` retournera la valeur de la température en °C à partir de l'entier `N_ana` fourni à chaque cycle par la carte d'entrée. Il est également possible d'obtenir les dérivées première et seconde de la grandeur mesurée avec une fonction de dérivation filtrée.

L'unité d'organisation de programme est composée de trois nouveaux types utilisateurs :

— la structure `RbAnalogInputParam_typ` qui contient les paramètres spécifiques de l'unité logicielle : gain, option de calcul des dérivées et valeurs des constantes des filtres de dérivation,
— la classe `RbAnalogInputParam` hérite de la classe `RbParam` et implémente la fonction abstraite de vérification de la compatibilité des paramètres, et retourne l'un des nouveaux identifiants en cas d'erreur,
— la classe `RbAnalogInput` hérite de la classe `RbLConCParam` qui lui attribue le modèle de comportement level-controlled continuous (l'état `Done` n'est jamais atteint) avec une structure de paramètres à vérifier. La classe implémente les méthodes `RbEnabling()` à `RbError()` pour particulariser son fonctionnement dans les différents états de la machine d'états générique.

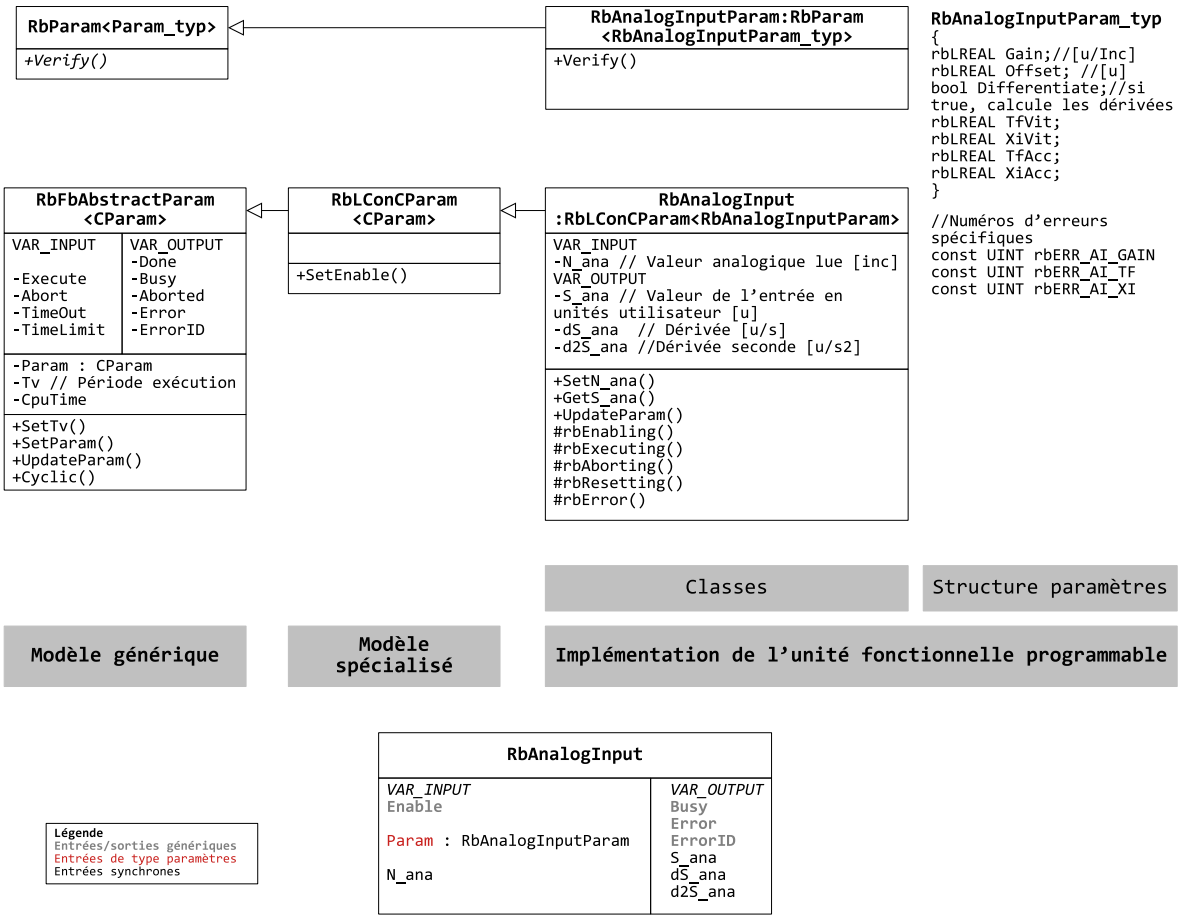


FIGURE 2.17 – Implémentation de l'unité d'organisation logicielle `RbAnalogInput` et bloc fonctionnel équivalent.

2.2.2 Axe de mouvement générique

La seconde fonctionnalité centrale de la bibliothèque `rtrmac` est la possibilité de contrôler de façon unifiée un grand nombre de solutions technologiques d'entraînement électrique. Cette propriété est obtenue en deux temps :

— l'ensemble des blocs fonctionnels du standard PLCopen Motion Control, partie 1 est exécuté au sein d'une unité d'organisation de programme qui coordonne l'action des blocs élémentaires, sélectionne les modes de pilotage utiles à un logiciel de pilotage robotique et fournit des interfaces permettant à d'autres unités fonctionnelles de prendre le contrôle des actionneurs,

— les interfaçages avec les différents types d'équipements de pilotage moteur sont réalisés par des classes spécialisées qui interagissent directement avec l'unité fonctionnelle de gestion d'axe de telle sorte que l'utilisateur n'ait pas besoin de prendre en compte la solution d'entraînement lorsqu'il développe son logiciel de pilotage. La prise en compte d'une architecture de contrôle particulière se résume au renseignement des paramètres fonctionnels utiles.

2.2.2.1 Abstraction du type d'entraînement électrique

Si l'on se place dans le cadre d'usage de systèmes d'entraînement avec une exigence de maîtrise du positionnement, il faut que l'ensemble formé du calculateur industriel et des cartes électroniques permettant l'adaptation de puissance pour la motorisation prenne en charge les fonctions suivantes :

— génération de la consigne instantanée de position à partir de la fonction choisie (déplacement vers une position d'arrêt absolue ou relative ou évolution vers une vitesse objectif),

— asservissement en position,

— asservissement en vitesse,

— asservissement en couple ou courant.

Suivant la topologie matérielle utilisée, ces différentes fonctions peuvent être réalisées par les préactionneurs ou rester à la charge du contrôleur principal. La figure 2.18 présente les différentes architectures matérielles qui peuvent être prises en charge par la bibliothèque *rtrmac*. Sur la figure, la topologie la plus basse correspond à l'utilisation d'un contrôleur de mouvement qui prend en charge l'essentiel de la génération de mouvement et n'échange avec l'automate que les ordres de haut niveau définis par l'usage des blocs fonctionnels PLCopen Motion Control. En conséquence, plus les cartes de contrôle sont simples, plus la charge de travail du calculateur augmente.

Pour pouvoir transférer de plus en plus de fonctions au contrôleur de mouvement, la bibliothèque *rtrmac* implémente la structure de régulation en cascade présente dans les contrôleurs de mouvement industriels présentée figure 2.19. Les suffixes `_ref` `_cons` et `_feed` font référence aux grandeurs de référence (issue de la génération de trajectoire), de consigne et d'anticipation, respectivement. Si une partie des boucles de régulation sont réalisées par les composants matériels (boucles de courant et de vitesse, par exemple dans le cas d'un variateur de vitesse), l'automate réalise le reste du schéma de régulation.

Ces quatre topologies sont complétées par une configuration de simulation pour laquelle la régulation est supposée parfaite et la position atteinte est constamment égale à la position de

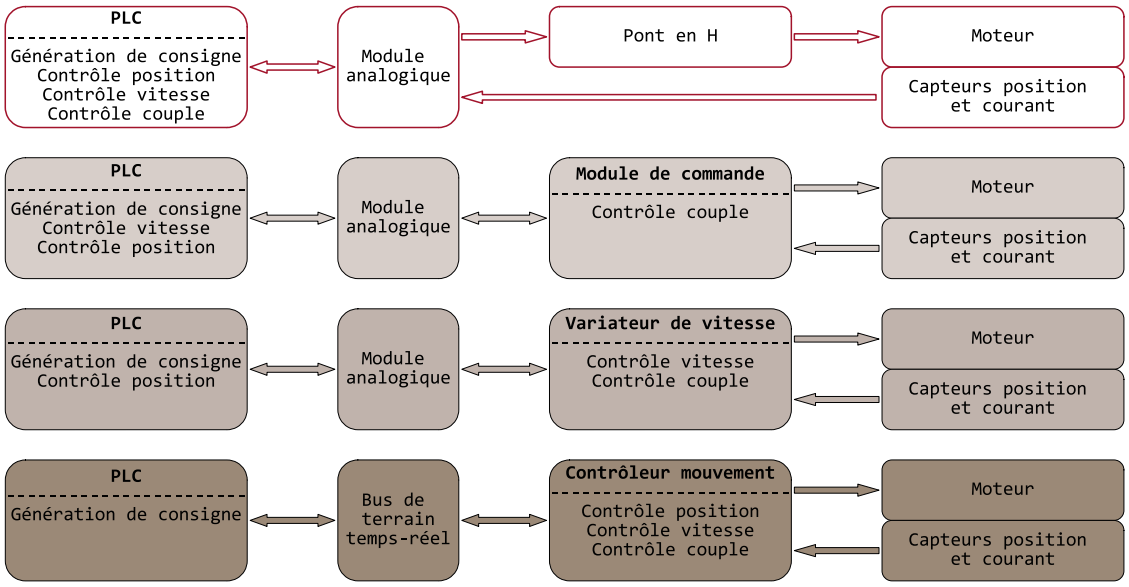


FIGURE 2.18 – Topologies d'axes asservis en position et répartition des fonctions suivant les architectures.

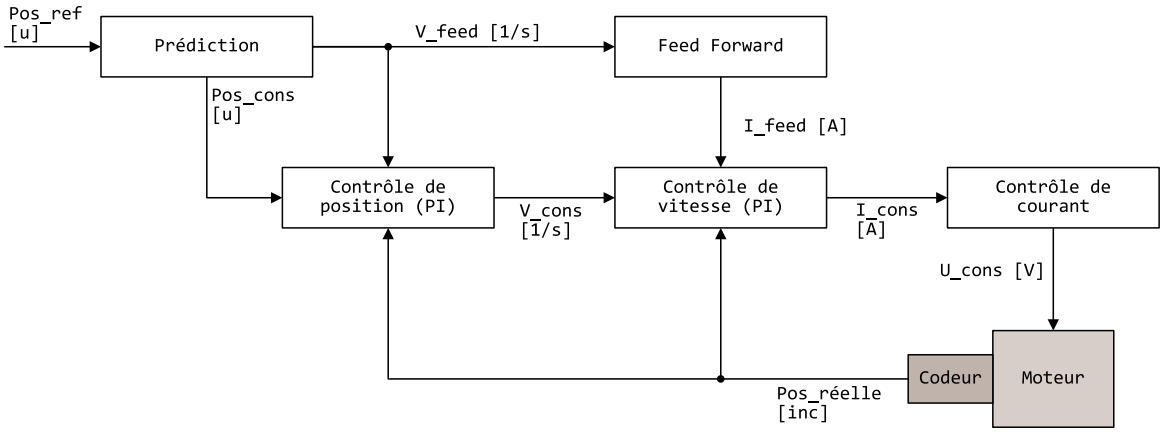


FIGURE 2.19 – Schéma simplifié de l'architecture de régulation en position des axes électriques, d'après *B&R Automation Help*.

consigne. Cette option est très importante dans le développement de la bibliothèque `rtrmac` puisqu'elle va permettre :

— de réaliser en avance et de tester le logiciel de commande d'une application pour laquelle la configuration matérielle n'est pas encore connue,
— de construire un axe virtuel qui sera contrôlé de façon identique aux axes réels,
— et de contrôler des axes ou des systèmes robotiques externes pour lesquels l'ensemble de la régulation est réalisée par la baie de commande du robot.

L'unification de ces différentes topologies est réalisée dans un objet logiciel qui contient les classes `RbMcAxe`, `RbMcAxeConfigParam` et la structure `RbMcAxeConfigParam_typ`, comme présenté sur la figure 2.20. Il traduit la topologie d'axe virtuel (c'est-à-dire que la régulation est parfaite) et peut être dérivé pour mettre en place les schémas de régulation associés aux différentes topologies. Cet objet logiciel implémente également la machine d'états PLCopen Motion Control (MC) de la figure 2.11. Pour toutes les topologies, des classes complémentaires permettent de simuler le comportement des parties opératives.

La structure `RbMcAxeConfigParam_typ` contient des éléments communs à toutes les topologies et des sous-structures associées aux différents composants du schéma de régulation qui ne sont nécessaires que dans certaines topologies.

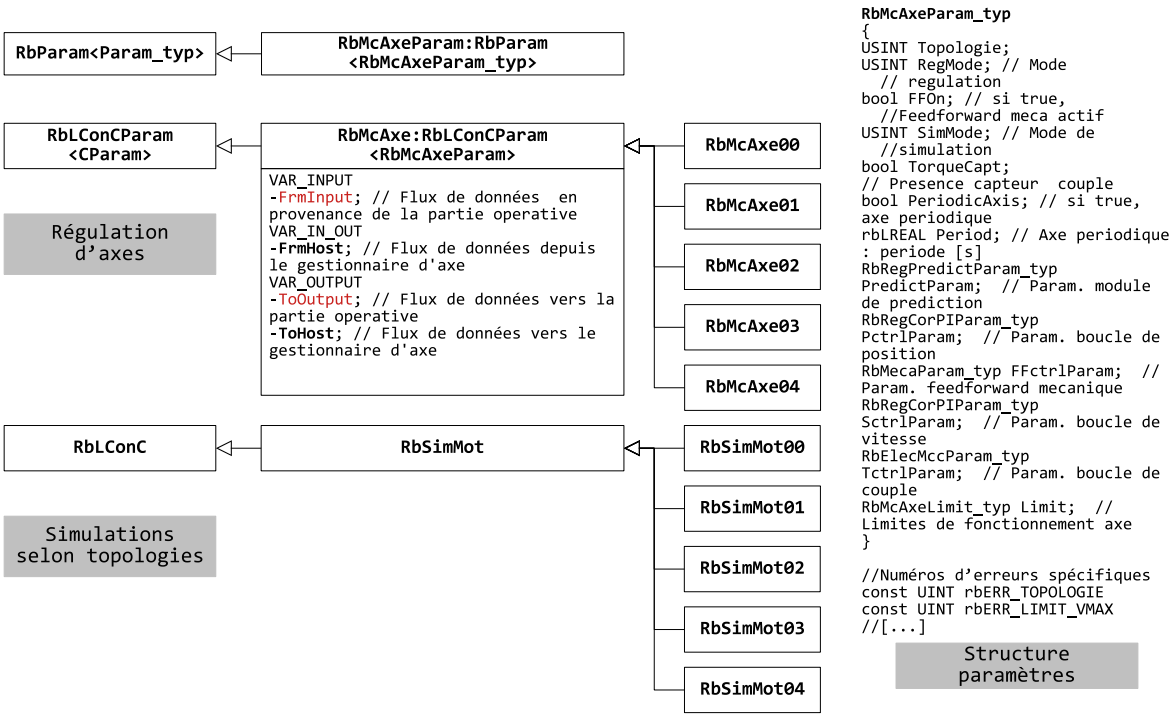


FIGURE 2.20 – Classes participant à la régulation d'axe.

Une sélection des blocs fonctionnels du standard est réalisée sous la forme de classes abstraites. L'exécution de ces blocs va contrôler l'axe piloté et faire évoluer son état dans la machine d'états PLCopen MC. Ils sont ensuite dérivés pour être utilisés avec les topologies d'axes inclues dans *rtrmac* ou pour encapsuler les blocs fonctionnels de bibliothèques propriétaires dans le cas de l'usage de contrôleurs de mouvement industriels (cf. figure 2.21). Ces blocs fonctionnels vont être utilisés pour construire l'unité d'organisation de programme qui va permettre de gérer l'axe

robotique présenté dans le chapitre suivant.

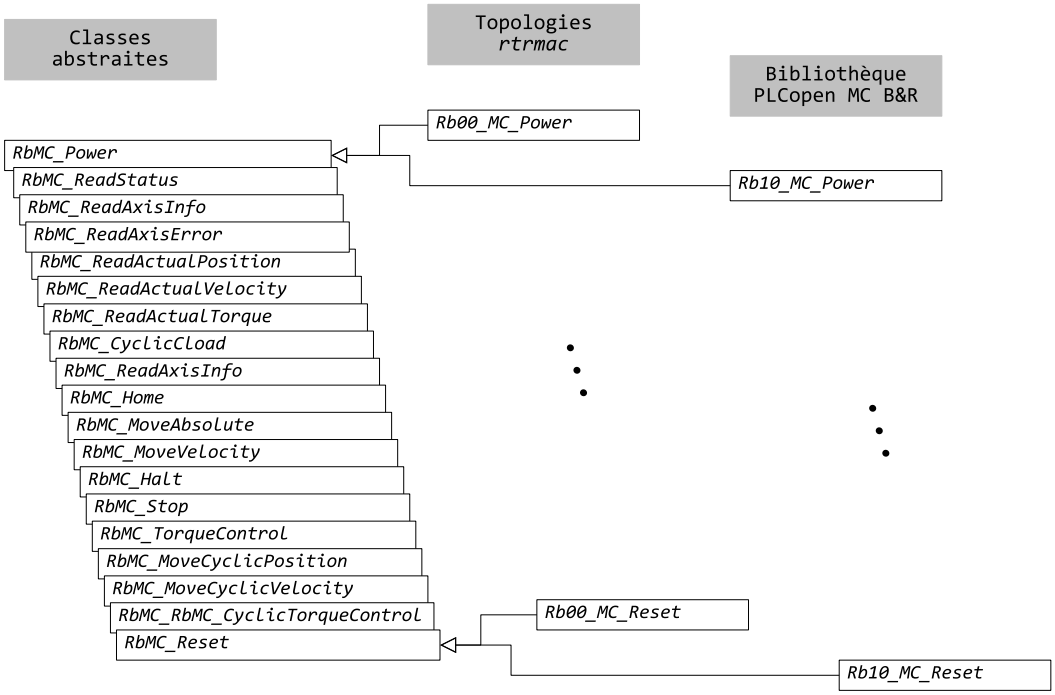


FIGURE 2.21 – Classes exécutant les comportements des blocs fonctionnels PLCopen Motion Control partie 1.

Ces blocs fonctionnels seront exécutés de façon masquée pour l'utilisateur par les gestionnaires d'axe robotique qui vont fournir une interface unique pour contrôler toutes les architectures matérielles ainsi que les axes de simulation.

2.2.2.2 Exemple de programme de régulation en position d'un actionneur électrique

Les figures 2.22 et 2.23 présentent les classes et structures utilisées dans l'implémentation de l'étage de régulation d'un axe électrique de topologie 1 : pont en H, module codeur et moteur courant continu avec codeur. Ces objets sont exécutés dans le programme `Hardware` qui est attribué à la classe de tâche prioritaire `Cyclic#1`. La première figure présente les actions menées pendant la phase d'initialisation de la tâche. L'exécution de la méthode `Enabling()` des classes de contrôle commande (dans la colonne de droite) charge les paramètres définis dans les structures après vérification de leur cohérence par les classes de vérification.

La figure 2.23 décrit le flux de données qui circule lors de l'exécution périodique des méthodes `Executing()`. La carte d'entrée de comptage fournit un nombre d'incréments qui est l'image de la position moteur. L'instance de la classe `RbCodeur` convertit cette valeur en position en unité de travail de l'utilisateur (en degrés, ici) et calcule les dérivées première et seconde. Ces trois informations servent d'entrées à l'objet `RegMot` qui instancie la classe de régulation des axes de

topologie 1. La sortie de ce bloc de régulation est convertie en entier de commande de la carte de puissance par l'instance de la classe `RbAnalogOutput`.

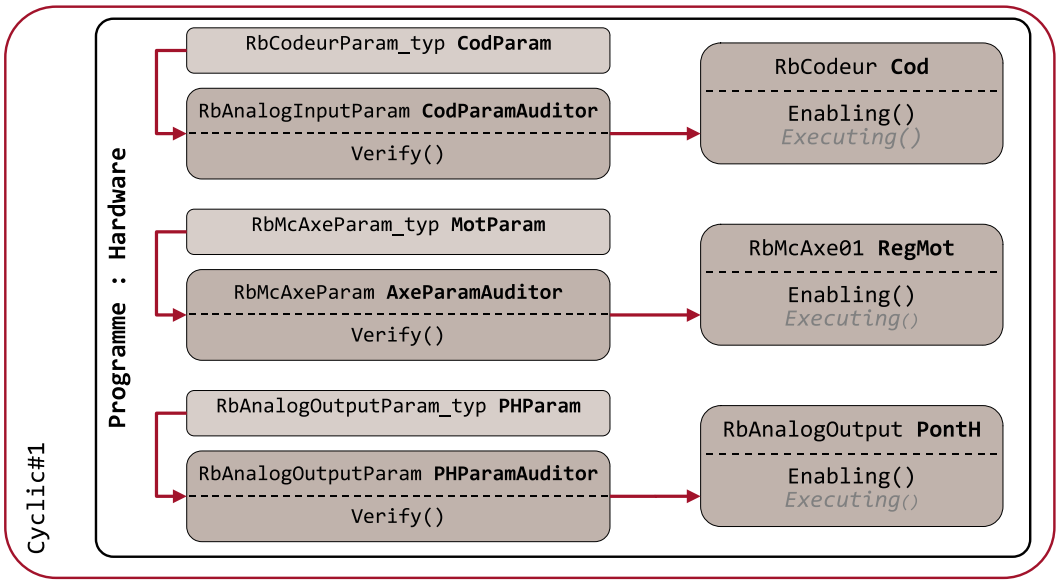


FIGURE 2.22 – Implémentation d'un programme de régulation moteur : chargement et vérification des paramètres à l'exécution des méthodes `Enabling()`.

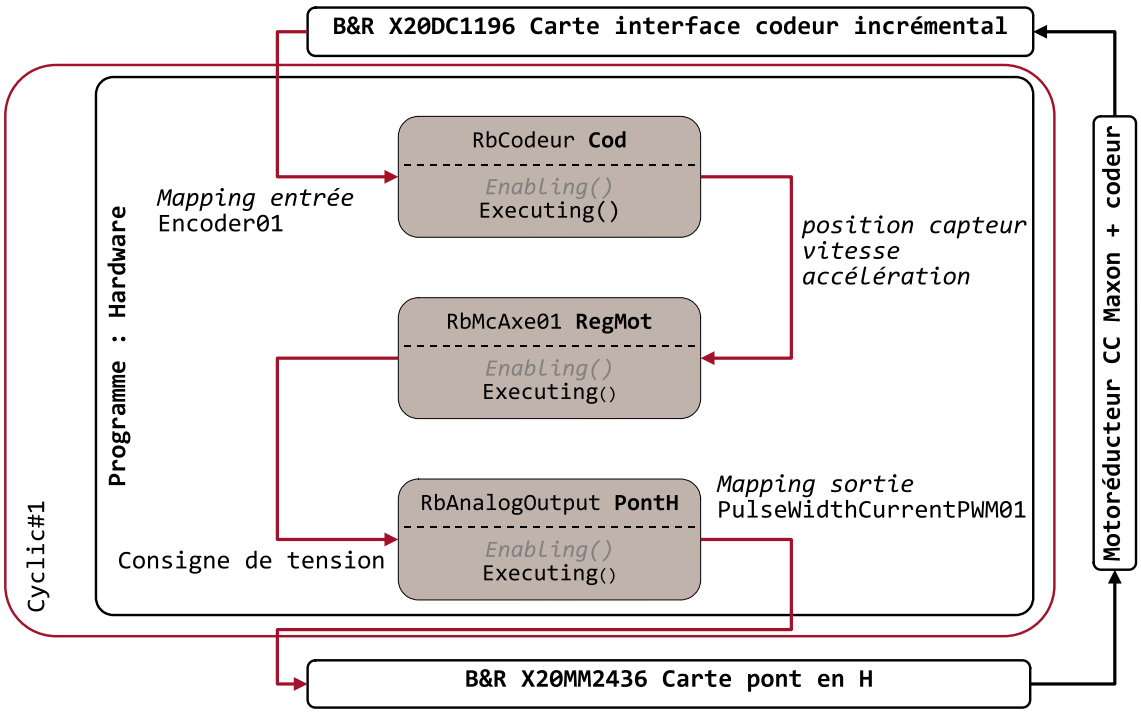


FIGURE 2.23 – Implémentation d'un programme de régulation moteur : flux de données dans la chaîne de traitement des entrées-sorties.

2.2.2.3 Réalisation d'un gestionnaire d'axe robotique

Pour simplifier l'usage des blocs fonctionnels PLCopen Motion Control, la bibliothèque *rtrmac* fournit une unité d'organisation logicielle permettant d'utiliser un composant appelé gestionnaire d'axe. Il donne accès aux blocs d'administration et de mouvement par une structure de commande, aux paramètres de l'axe, par une structure de paramètres et aux informations sur l'état de l'axe par une structure d'information. Cette fonctionnalité est fournie, par certains fournisseurs de matériel, sous la forme de blocs fonctionnels propriétaires puisque ce niveau d'intégration n'est pas imposé par le standard de commande d'axe.

L'unité d'organisation logicielle de gestion d'un axe robotique donne un accès indirect aux blocs fonctionnels de commande d'axe par l'intermédiaire d'une structure de commande et d'une structure d'information qui pourront être échangées entre programmes. La modification de la structure de commande permet d'activer et de passer les paramètres des différentes fonctions de mouvement nécessaires au pilotage de mécanismes et de robots. Ces fonctions seront accessibles pour le pilotage des moteurs et des articulations des robots ou des sorties des mécanismes. Il est possible de choisir un mode de contrôle (en position, en vitesse ou en couple, selon les possibilités offertes par le composant d'adaptation de puissance). Pour chacun, le gestionnaire d'axe peut générer une loi de commande vers un objectif à atteindre ou suivre une consigne qui sera fournie périodiquement par un générateur externe au gestionnaire d'axe.

Comme pour l'étage de régulation présenté dans le chapitre précédent, l'implémentation est réalisée sous la forme de classes abstraites à dériver pour contrôler les différentes topologies d'axes pris en charge et pour s'interfacer avec les bibliothèques de commande d'axe des fournisseurs de matériel.

L'unité d'organisation logicielle abstraite réunit les classes `RbMcAxeRob` et deux structures d'échange, l'une contient les ordres utilisateurs permettant de déclencher des actions de préparation ou de mouvement (démarrage, arrêt, déplacement vers une position ou une vitesse objectifs, par exemple), la seconde retourne des informations sur l'état de l'axe qui seront exploitées par le programme de niveau hiérarchique supérieur : le gestionnaire de groupe d'axes ou le gestionnaire machine.

Cette classe abstraite exécute des instances des classes représentant les blocs fonctionnels PLCopen partie 1 présentés au paragraphe précédent et coordonne leur usage par une machine d'états qui définit à chaque situation les actions disponibles. Elle est dérivée (cf. figure 2.24) pour s'adapter à différents usages :

- la classe `Rb00_MC_AxeRob` permet de contrôler les différentes topologies d'entraînements électriques,
- la classe `RbJoint` va permettre de traiter de façon homogène aux axes moteurs les axes articulaires des systèmes robotique, ce sont des axes parfaits, du point de vue de la régulation,
- et la classe `Rb10_MC_AxeRob` permet de prendre en charge les contrôleurs de mouvement de la marque B&R Automation.

La figure 2.25 présente la machine qui coordonne les actions de démarrage et arrêt des étages de puissance, de prise d'origine et d'arrêt d'urgence logiciel. Cette dernière fonctionnalité interrompt toutes les consignes de génération de mouvement et réalise un freinage contrôlé jusqu'à l'arrêt de l'axe.

Lorsque l'axe est dans l'état `Standstill`, l'utilisateur peut choisir l'un des modes de pilotage

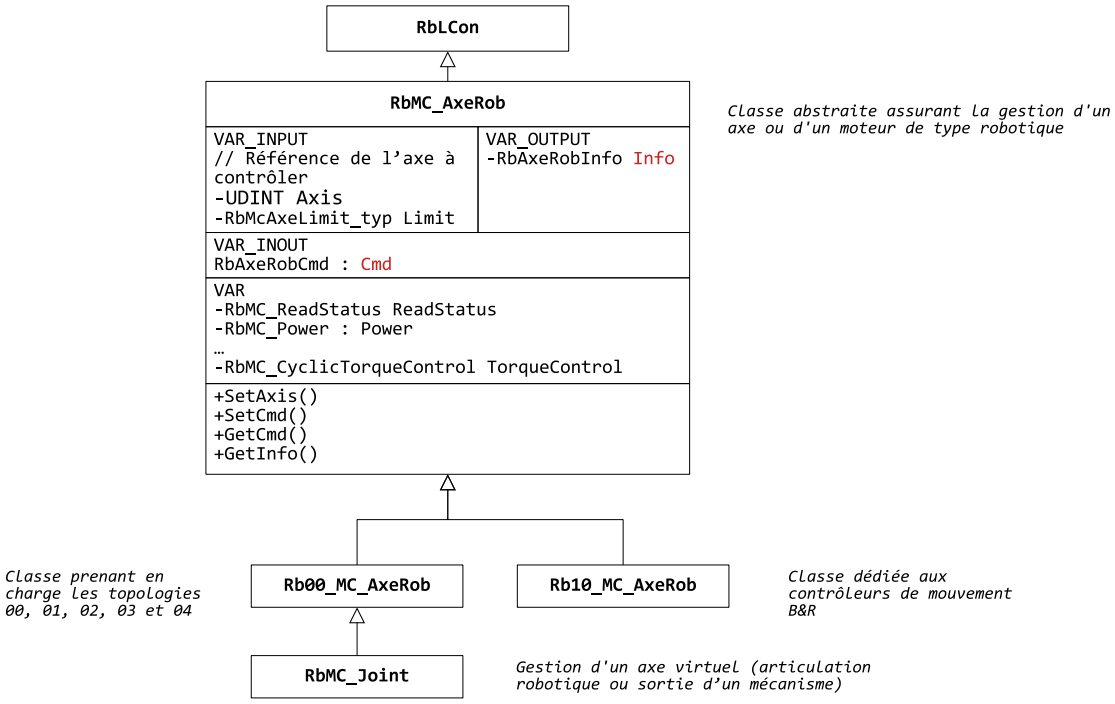


FIGURE 2.24 – Classes implémentant les gestionnaires d'axes robotiques.

présentés sur la figure 2.26. Le gestionnaire de mouvement donne accès à un mode manuel de type *Jogging* qui est un déplacement à vitesse constante tant que la commande de mouvement est active, un déplacement vers un point d'arrêt et à plusieurs modes de suivi de consigne périodique. Dans ce dernier cas, la variation de consigne est générée à l'extérieur du gestionnaire d'axe, à la période d'exécution de l'automate. Il peut s'agir d'une consigne :

— de position ou de vitesse,

— de couple, si le préactionneur a un accès à la boucle de régulation en courant.

Cette partie de la bibliothèque *rtrmac* a été testée intensivement et est une brique élémentaire présente dans les logiciels de pilotage de tous les dispositifs de l'équipe RoBioSS. Elle sera utilisée dans les exemples de mécanismes et de robots présentés dans les parties 3 et 4 de ce manuscrit.

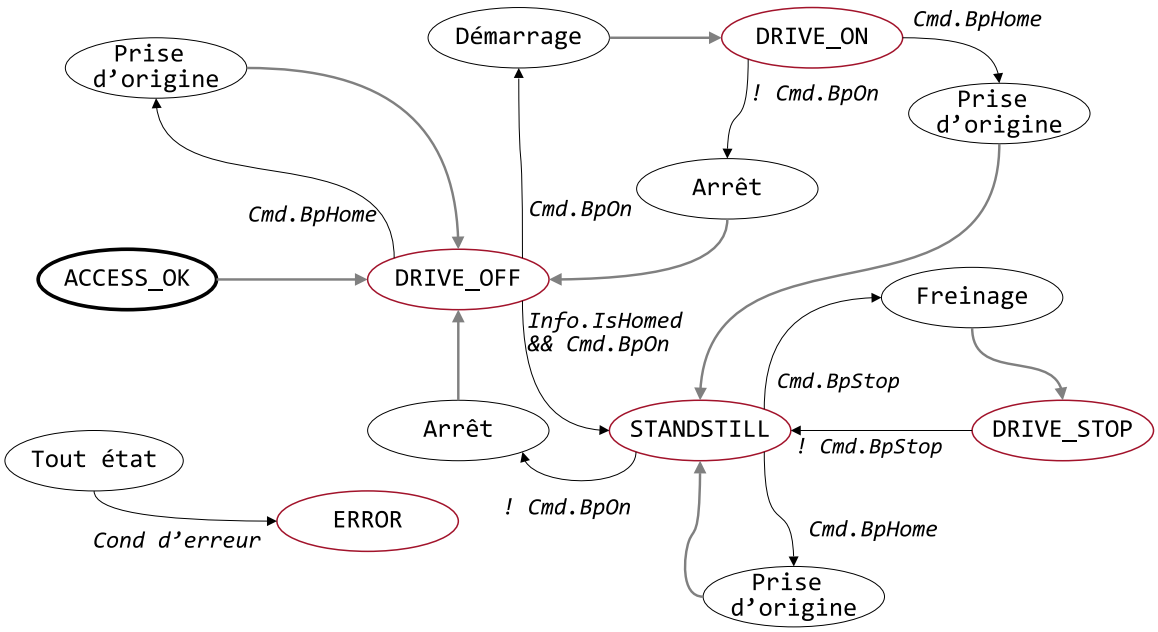


FIGURE 2.25 – Axe robotique : étapes de préparation avant mouvement (démarrage - arrêt, prise d'origine, arrêt d'urgence logiciel).

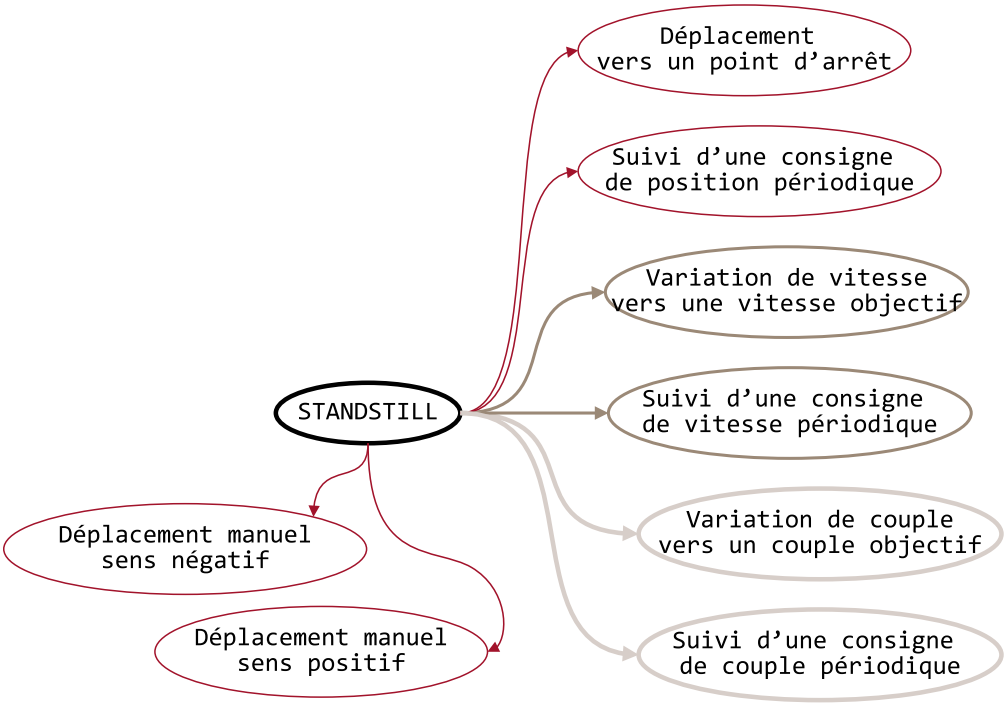


FIGURE 2.26 – Modes de pilotage d'un axe robotique.

2.2.2.4 Démarche d'extension de la bibliothèque

Ce travail de mise en place du standard PLCopen MC a pour objectif de pouvoir également intégrer des actionneurs entraînés par des contrôleurs de mouvement industriels. La bibliothèque *rtrmac* fournit la possibilité d'utiliser la gamme B&R Automation par l'encapsulation de la bibliothèque propriétaire qui fournit l'implémentation du standard pour ce matériel, en dérivant les dix-huit classes présentées précédemment et le gestionnaire d'axe robotique. Cette démarche sera à reproduire pour l'ajout de nouvelles gammes de matériels de fournisseurs différents.

2.3 Une démarche d'unification du développement logiciel

Pour améliorer la performance globale d'un système mécatronique, il est nécessaire que la prise d'informations (la phase *Sense* du schéma sense - plan - act, figure 1.4) se fasse de façon synchrone avec le traitement de l'information. Le choix d'exploiter des composants industriels permet d'avoir accès à une très large gamme de capteurs et de cartes d'acquisitions qui échangent des informations avec le contrôleur central à une période d'échange maîtrisée.

Il est possible d'utiliser la bibliothèque *rtrmac* pour développer des interfaces logicielles pour l'acquisition de données de capteurs spécifiques. Dans cette section, nous en présentons deux exemples : le premier montre comment permettre à l'utilisateur d'accéder à des fonctions intégrées dans les cartes d'acquisition, le second présente la démarche de création d'un bloc fonctionnel associé à un capteur spécifique en agrégeant des informations composites.

2.3.1 Usage optimal des composants matériels

La bibliothèque `rtrmac` fournit une brique de base permettant la conversion d'une entrée analogique en image de la grandeur mesurée (cf. figure 2.17). Les fonctions présentes dans la classe de base `RbAnalogInput` sont les suivantes :

— conversion de la variable entière associée à la conversion numérique analogique de la carte en variable réelle ,
— réglage d'offset,
— possibilité de dériver la grandeur mesurée.

Cette classe peut être utilisée pour développer de nouvelles interfaces pour exploiter des capteurs spécifiques ou être dérivée pour rajouter de nouvelles fonctions. En particulier, pour mesurer les interactions entre un système mécatronique et son environnement, nous sommes amenés à mettre en œuvre des capteurs de force multiaxes. Il est également souvent nécessaire de filtrer les entrées analogiques. Cette fonction de filtre dépend de la dynamique du capteur, de celle du contrôle-commande à réaliser et aura un impact sur la résolution finale de l'information mesurée. Les cartes d'acquisition permettent de réaliser certaines opérations d'amélioration du signal. Dans la logique d'exploitation de la capacité de puissance d'un système distribué, il est intéressant de pouvoir exploiter au mieux les fonctions fournies par les unités de traitement.

2.3.1.1 Composant exploitant des fonctions embarquées dans la carte d'acquisition

Pour réaliser des mesures d'actions mécaniques qui transitent dans des mécanismes ou des chaînes de transmission de mouvement, nous utilisons régulièrement des jauges de déformation, intégrées dans des capteurs ou placées sur des corps d'épreuve dans les mécanismes que nous

concevons. Les cartes d'acquisition B&R Automation qui font la mesure et l'amplification par pont de Wheatstone disposent également de filtres numériques évolués (cf. 2.27).

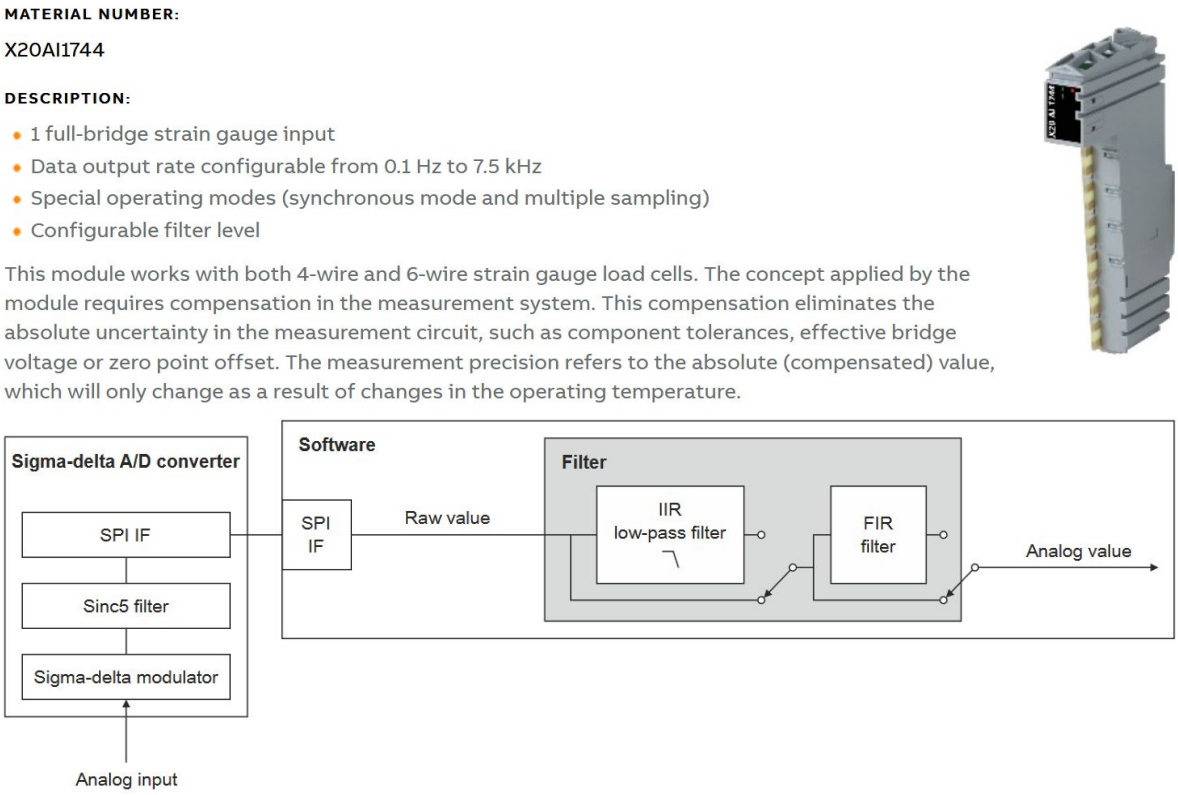


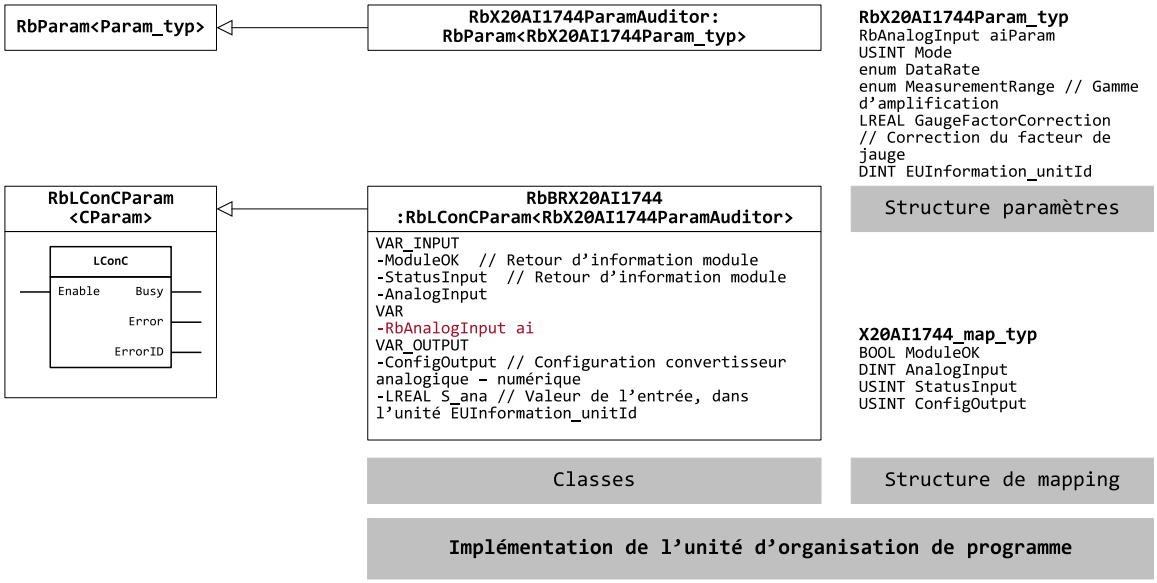
FIGURE 2.27 – Extrait de la présentation de la carte d'acquisition B&R X20AI1744. Caractéristiques générales et filtres implémentables.

Dans une logique d'utilisation optimale des ressources de calcul disponibles et de traitement au plus près de la mesure, il est intéressant d'exploiter au mieux les fonctions intégrées dans les cartes d'entrées analogiques.

Pour cela, il est possible de réaliser des blocs fonctionnels dédiés, associés à des cartes d'entrées spécifiques, qui permettent d'offrir à l'utilisateur des interfaces permettant de contrôler les fonctions prises en charge par la carte.

La carte B&R X20AI1744 permet d'amplifier et de convertir en valeur numérique la mesure de déformation donnée par le changement de résistance d'une jauge de déformation. L'amplification est définie par le facteur de jauge nominal et la carte permet d'effectuer plusieurs types de filtrage numérique. Le paramétrage du facteur d'amplification et le réglage des caractéristiques des filtres sont modifiables de façon synchrone par l'applicatif utilisateur.

Il est possible de créer une unité logicielle de programme associée à la carte d'entrée (cf. figure 2.28) à partir de la fonction élémentaire de traduction d'une entrée analogique en variable représentant une grandeur physique. Ce nouveau composant logiciel donne accès à l'utilisateur aux fonctions intégrées dans la carte à travers son interface d'entrée. Il ajoute également une description de la nature et de l'unité de la grandeur physique mesurée, dans l'esprit d'OPC-UA :



Channel Name	Process Variable	Data Type	Description [1]
ModuleOk	::Hardware:X20AI1744.ModuleOk	BOOL	Module status (1 = module present)
SerialNumber		UDINT	Serial number
ModuleID		UINT	Module ID
HardwareVariant		UINT	Hardware variant
FirmwareVersion		UINT	Firmware version
AnalogInput01	::Hardware:X20AI1744.N_ana	DINT	DMS value
AdcConvTimeStamp01		DINT	Timestamp of last AD conversion
StatusInput01	::Hardware:X20AI1744.StatusInput01	USINT	Module status
ConfigCommonOutput01	::Hardware:X20AI1744.ConfigCommonOutput01	USINT	ADC configuration
ConfigDatarateOutput01	::Hardware:X20AI1744.ConfigDararateOutput01	USINT	Module config baudrate

FIGURE 2.28 – Unité d'organisation de programme dédiée à la carte B&R X20AI1744 et table de mapping avec les entrées/sorties de la carte.

le champ `EUInformation_unitId` de la structure de paramètres est renseigné avec la référence de cette grandeur définie dans la recommandation *Codes for Units of Measurement used in the International Trade* émise par le *United Nations Centre for Trade Facilitation and Electronic Business* [UNECE 2022]. Cette information peut être utilisée pour effectuer des conversions d'unités ou pour simplifier la création d'une table d'échange d'un serveur OPC-UA.

2.3.1.2 Composant intégrant plusieurs capteurs élémentaires

Dans le même principe d'améliorer la phase de prise d'informations, nous avons réalisé un bloc fonctionnel associé à la mesure d'une information composite obtenue à partir de plusieurs entrées analogiques.

Nous utilisons, au sein de l'équipe RoBioSS, des capteurs d'efforts multiaxes de la société Sensix pour des applications biomécaniques ou robotiques. Ils se présentent avec plusieurs facteurs de forme et des gammes de mesure très variables. La figure 2.29 en présente trois exemples : une plateforme de force exploitée pour déterminer les actions mécaniques exercées sur le sol par

un sujet ou un robot humanoïde et des capteurs d'efforts six composantes à intégrer dans des ergomètres, des dispositifs mécatroniques d'assistance, ou des cellules de cobotiques. Tous ces capteurs sont basés sur l'exploitation d'un groupe de jauges de déformations. Les capteurs délivrent une série de signaux analogiques amplifiés qui permettent d'obtenir le torseur d'actions mécaniques exercé sur le capteur à partir d'une matrice de calibration.

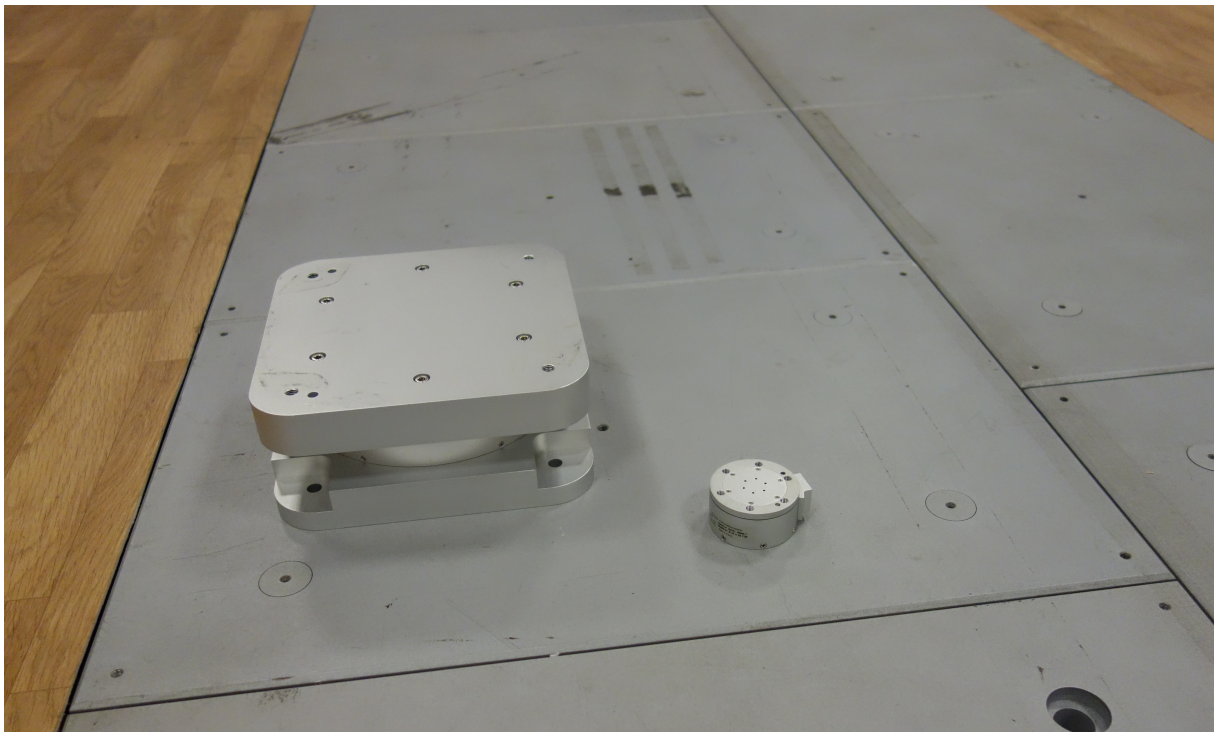


FIGURE 2.29 – Exemples de capteurs d'actions mécaniques Sensix, capteurs six axes et plateformes de force disposées en chemin de marche.

L'unité d'organisation de programme présentée figure 2.30 offre une interface unique pour l'ensemble de nos capteurs d'actions mécaniques. Ce composant logiciel est en cours de développement pour ajouter des fonctions de filtrage numérique des signaux générés similaires à celles de la carte analogique présentée précédemment. L'objectif de ce travail est de disposer d'une interface identique sur tous les blocs fonctionnels de gestion des entrées analogiques, les fonctions de filtrages étant prises en charge par la carte d'acquisition ou par l'automate, suivant les configurations matérielles utilisées.

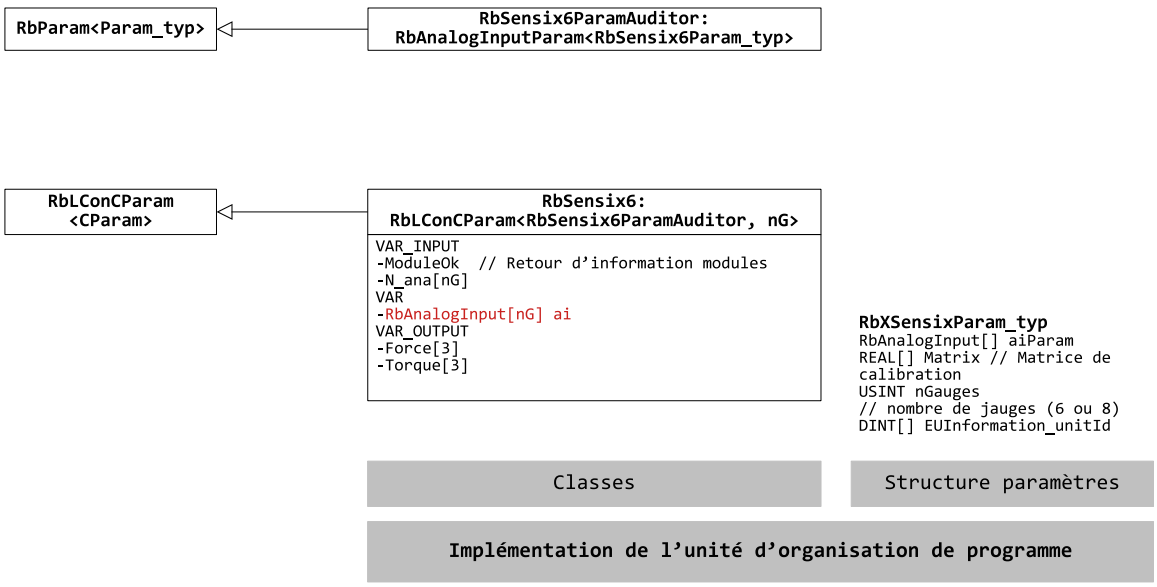


FIGURE 2.30 – Bloc fonctionnel de capteurs d'actions mécaniques six composantes.

2.3.2 Mise en œuvre d'algorithmes complexes

Dans notre stratégie de développement logiciel, l'exigence de pouvoir intégrer du code réalisé en C++ permet d'intégrer des bibliothèques tierces. Il est cependant nécessaire que ces dernières soient conçues pour un usage sur une cible temps-réel, qu'elles ne requièrent pas elles-même de nouvelles dépendances et qu'elles ne fassent pas usage d'allocation dynamique de mémoire dans la fonction exécutée périodiquement.

Lorsque ces conditions sont réunies, il est possible d'intégrer de nouvelles fonctions utiles sous la forme de classes fonctionnant comme des blocs fonctionnels. A titre d'exemple, je présente l'ajout d'une classe d'interpolation à partir de la bibliothèque *Reflexxes*. Cette classe est utilisée dans l'architecture de contrôle commande multiaxe présentée au chapitre 3.

Les bibliothèques Reflexxes permettent de générer en ligne des lois de commande de mouvement dont la destination peut varier de façon dynamique [Kröger 2011] [Kröger 2010]. Nous avons intégré la version `II` de cette bibliothèque qui permet de calculer à chaque instant une nouvelle consigne `(positions, vitesses)` pour un ensemble de composantes de vecteurs déplacements, en connaissant pour l'instant courant, les vitesses et positions actuelles, les limites en vitesses et en accélérations et la destination de ces mouvements.

L'exemple suivant illustre un cas d'utilisation de la méthode d'interpolation de consigne de position (`RMLPosition`) de la classe `ReflexxesAPI` de la bibliothèque `Reflexxes type II`. Cette méthode a été intégrée dans une unité d'organisation logicielle qui se comporte comme un bloc fonctionnel du standard IEC61131-3. Le bloc fonctionnel équivalent est présenté sur la figure 2.32 avec les structures qui lui sont associées.

Un programme C++ exécute périodiquement la méthode `Cyclic()` d'une instance de cet interpolateur. Ce programme correspond à un déplacement vers une destination absolue,

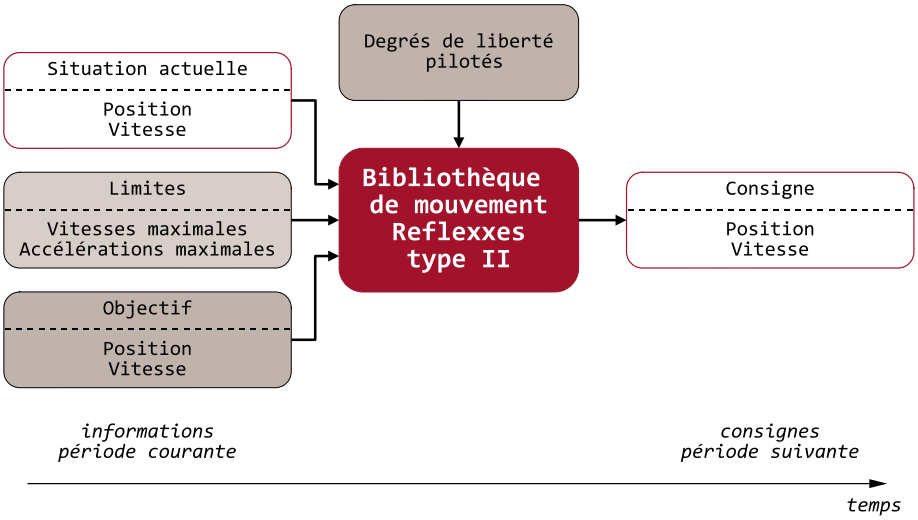
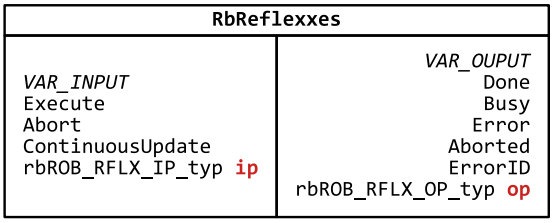


FIGURE 2.31 – Génération de consigne de position, *Reflexxes type II*, d'après [Kröger 2011] .

définie à l'activation de la variable `Execute`. Le mouvement prend fin lorsque la méthode `ReflexxesAPI::RMLPosition()` retourne l'information que l'algorithme de génération de trajectoire a atteint la cible prévue. Les entrées et sorties de l'interpolateur sont copiées dans une variable d'échange partagée avec le reste des programmes.

Ceci permet à un programme utilisateur d'exploiter la bibliothèque *Reflexxes* avec la même interface et le même comportement qu'un bloc fonctionnel compatible avec les préconisations de développement émises par PLCopen.

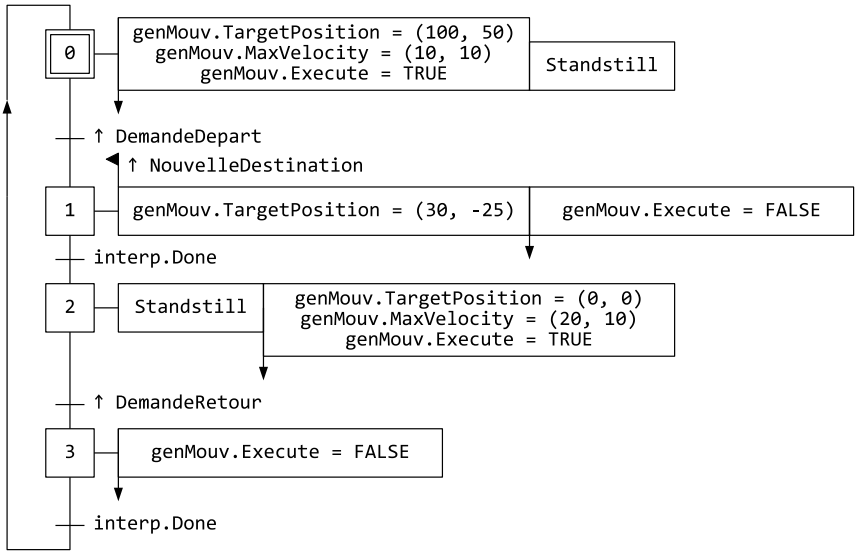
La figure 2.33 présente la spécification et le programme en Texte Structuré qui la traduit tels qu'un technicien de bureau d'études pourrait les réaliser. La fonction d'interpolation est utilisable par un professionnel de l'automatisme sans exigence de compétence complémentaire en développement informatique ou en algorithmique.

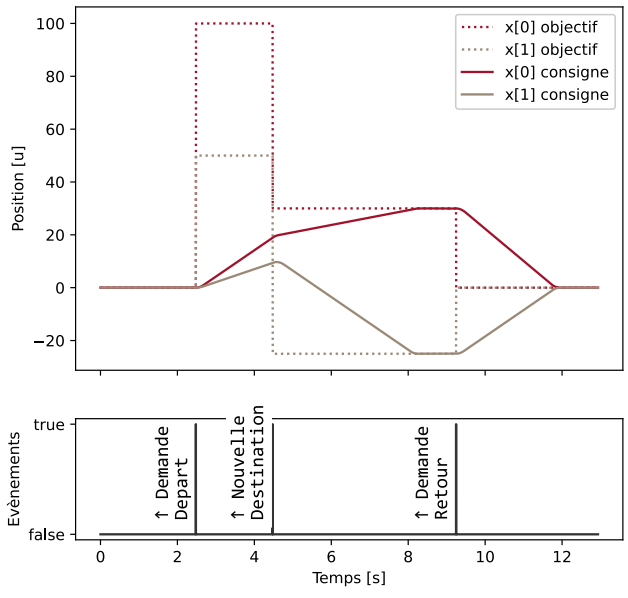



```
rbROB_RFLX_IP_typ
{
double CurrentPositionVector[N_AXES];
double CurrentVelocityVector[N_AXES];
double
CurrentAccelerationVector[N_AXES];
double MaxVelocityVector[N_AXES];
double MaxAccelerationVector[N_AXES];
double MaxJerkVector[N_AXES];
double TargetPositionVector[N_AXES];
double TargetVelocityVector[N_AXES];
bool SelectionVector[N_AXES];
}

rbROB_RFLX_OP_typ
{
double NewPositionVector[N_AXES];
double NewVelocityVector[N_AXES];
double NewAccelerationVector[N_AXES];
}
```

```
Structure d'interface entre le programme C++
et des programmes IEC61131

RbReflexxes_typ
{
bool Execute;
bool Abort;
bool ContinuousUpdate;
rbROB_RFLX_IP_typ ip;
bool Busy;
bool Done;
bool Aborted;
bool Error;
UINT ErrorID;
rbROB_RFLX_OP_typ op;
double Period;
}
```

FIGURE 2.32 – Interpolation en position Reflexxes : bloc fonctionnel équivalent.






```
PROGRAM _CYCLIC

CASE Etat OF
  0:
    IF DemandeDepart THEN
      DemandeDepart := FALSE;
      Etat := 1;
      genMouv.Execute := TRUE;
      genMouv.ip.TargetPositionVector[0] := 100.0;
      genMouv.ip.TargetPositionVector[1] :=  50.0;
    END_IF
  1:
    IF genMouv.Done THEN
      genMouv.Execute := FALSE;
      Etat := 2;
    ELSIF NouvelleDestination THEN
      NouvelleDestination := FALSE;
      genMouv.ip.TargetPositionVector[0] :=  30.0;
      genMouv.ip.TargetPositionVector[1] := -25.0;
    END_IF
  2:
    IF DemandeRetour THEN
      DemandeRetour := FALSE;
      Etat := 3;
      genMouv.ip.TargetPositionVector[0] := 0.0;
      genMouv.ip.TargetPositionVector[1] := 0.0;
      genMouv.ip.MaxVelocityVector[0]  := 20.0;
      genMouv.Execute := TRUE;
    END_IF
  3:
    IF genMouv.Done THEN
      genMouv.Execute := FALSE;
      Etat := 0;
    END_IF
END_CASE
```

FIGURE 2.33 – Spécification GRAFCET d'un exemple d'usage de l'interpolateur en position *Reflexxes*, code automate en ST et chronogrammes des déplacements générés.

2.4 Conclusion

La bibliothèque *rtrmac* fournit des classes abstraites permettant d'attribuer aux classes dérivées un comportement homogène et régulier pour des composants logiciels répondant à des fonctions de toute nature, mais qui s'exécutent de façon périodique sur un système d'exploitation temps-réel dur. La bibliothèque dispose d'un ensemble d'outils nécessaires à la réalisation des fonctions d'intégration des composants matériels : adaptation des signaux d'entrée, régulation, gestion d'axes de mouvement quelle que soit la technologie d'entraînement électrique utilisée.

La structure d'une unité d'organisation de programme permet d'activer ou d'arrêter la fonction qu'elle réalise avec une interface utilisateur identique aux blocs fonctionnels des languages automates. Elle offre un mécanisme de validation des paramètres et une gestion des erreurs qui permettent aux programmes qui l'exécutent de ne pas rencontrer d'erreur fatale et mettre en place des stratégies d'attente d'action corrective de l'utilisateur ou d'abandon de la fonction demandée.

rtrmac peut être étendue pour :

— rendre accessibles les fonctions intégrées dans les cartes d'entrée-sortie,

— simplifier l'interface de capteurs composites,

— encapsuler les fonctions algorithmiques avancées de bibliothèques tierces.

Ces composants logiciels peuvent être exploités dans des programmes C++ ou, par l'intermédiaire de structures d'interface, dans les programmes codés en langage du standard automate, ce qui rend leur usage accessible aux spécialistes du monde de l'automatisme.

Le chapitre suivant porte sur l'extension de la bibliothèque *rtrmac* à une couche fonctionnelle supérieure à la gestion d'axe : le contrôle de mécanismes et de robots.

Chapitre 3

Développement d'une architecture personnalisable de contrôle de systèmes mécaniques

Cette partie présente le travail réalisé pour l'extension de la bibliothèque *rtrmac* au pilotage d'une grande variété de systèmes.

Cette évolution part du besoin généré par l'activité de l'équipe RoBioSS qui conçoit de nombreux dispositifs mécatroniques qui sont majoritairement en situation d'interaction avec des agents humains ou qui évoluent en environnement incertain.

Les exigences rencontrées sont les suivantes :

— développer un nouveau système robotique ou un banc de test dans une durée compatible avec la réalisation d'une thèse ou d'un projet de recherche et avec des ressources en ingénierie limitées,
— contrôler des systèmes très différents en termes de nombre d'axes d'entraînement, de nombre de capteurs à intégrer, de nombre d'effecteurs,
— pouvoir exploiter des transmissions mécaniques complexes,
— associer sur le même projet des acteurs avec des objectifs différents et des compétences en développement informatique variées.

La figure 3.1 présente une vue panoramique du travail réalisé pour la conception de cette architecture de contrôle-commande multiaxe. La figure est partagée en trois zones.

La description de la configuration matérielle fait apparaître les composants matériels des cellules robotiques que nous sommes amenés à développer. Comme présenté dans le chapitre 1.2.3, il s'agit d'une architecture où le traitement de l'information est réparti entre plusieurs calculateurs présents dans différents composants (PLC ou PC industriels, automate de sécurité, cartes électroniques, contrôleurs de mouvement). Certaines fonctions s'exécutent en parallèle, à partir d'informations différentes. C'est le cas de l'automate et de l'automate de sécurité qui ont des responsabilités différentes. Les calculateurs industriels modernes permettent également de travailler de manière redondante pour garantir la continuité de fonctionnement de l'installation en cas de défaillance du calculateur central.

La couche robotique de la bibliothèque *rtrmac* est structurée autour d'une décomposition en plusieurs espaces de description des mouvements : les actionneurs, les articulations pilotées, l'ensemble des articulations (qui peut être différent du groupe précédent dans le cas de mécanismes sous-actionnés), les axes de mouvement dans l'espace opérationnel. Les mécanismes de préparation du robot (mise sous tension, prise d'origine, couplage moteurs - articulations) et de gestion de l'arrêt d'urgence logiciel sont des éléments essentiels qui structurent le fonctionnement des programmes de gestion des robots. Ces éléments sont présentés aux sections 3.3 et 3.4.

Enfin, pour une application donnée, le logiciel de contrôle-commande est décomposé en strates fonctionnelles dont :

— la responsabilité du développement peut être confiée à des acteurs différents
— l'exécution peut être affectée à un ou plusieurs calculateurs, selon la puissance de calcul disponible.

Ce point est détaillé à la section 3.2.2.

La première partie de ce chapitre présente les solutions industrielles disponibles ou en cours de développement pour le contrôle d'architectures robotiques. Je décris ensuite la démarche retenue pour la conception d'une nouvelle solution de contrôle générique de systèmes mécatroniques. Les deux parties suivantes portent plus précisément sur la définition des mécanismes et des robots. La dernière partie montre la mise en place de la démarche complète sur une architecture simple.

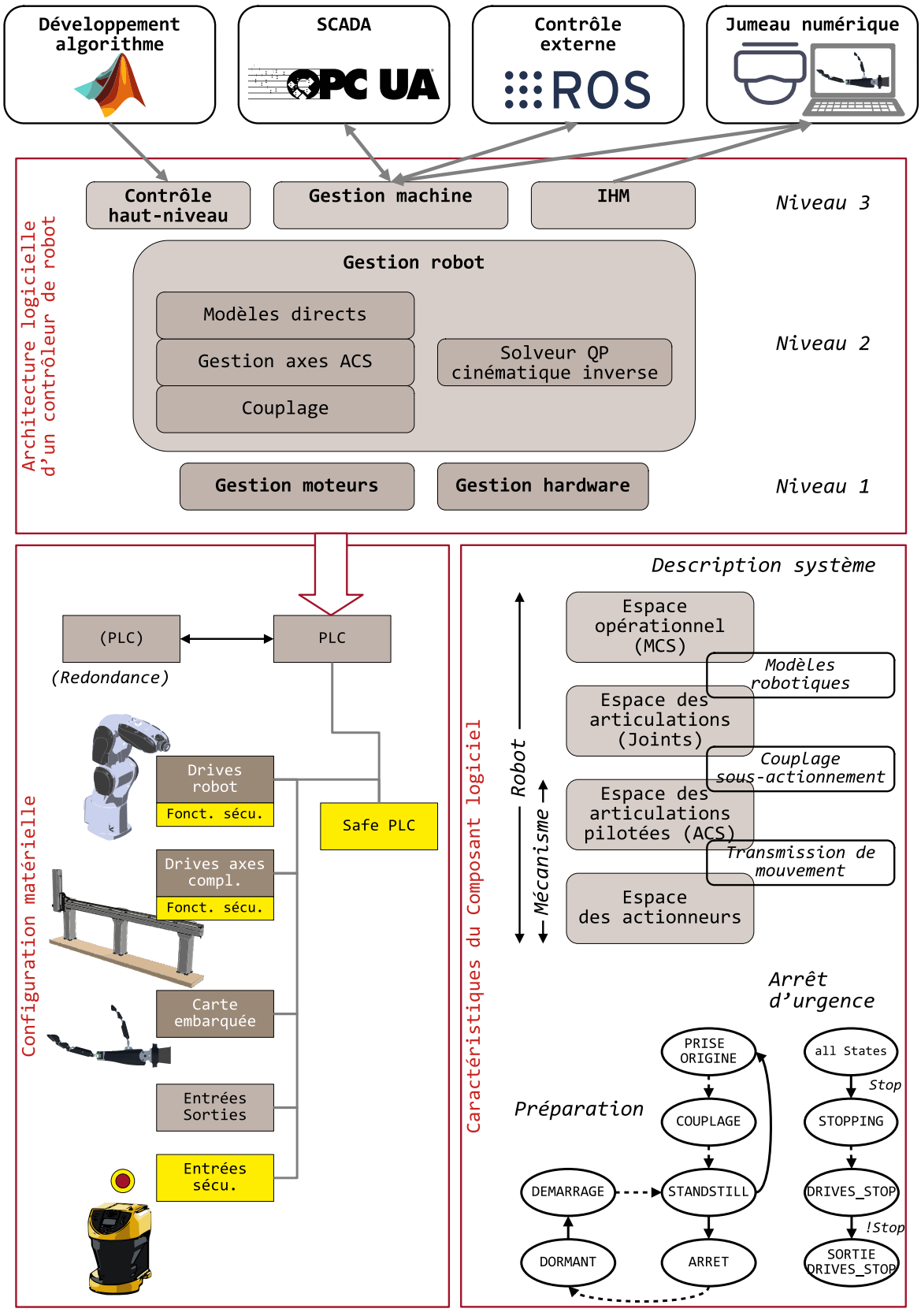


FIGURE 3.1 – Principe de réalisation d'un contrôleur de robot par extension de la bibliothèque *rtrmac*.

3.1 Contrôle commande multiaxe industriel : des approches antagonistes

Le domaine de la programmation de machines industrielles multiaxes comporte un très grand nombre d'acteurs qui développent des solutions centrées sur leur matériel ou leur service et qui cherchent à étendre leur usage à des fonctions périphériques.

Les acteurs historiques sont les fabricants de composants de contrôle-commande, en particulier d'automates et de contrôleurs de mouvements et les fabricants de robots qui fournissent des quasi-machines avec des suites logicielles très évoluées et des solutions d'assistance et de maintenance. D'autres acteurs sont des fabricants de machine OEM qui développent leurs propres solutions pour leurs gammes de machine, à partir de technologies d'automatismes ou de commandes numériques et il y a également des entreprises d'ingénierie logicielle qui développent des solutions de contrôle externes ou des environnements de simulation pouvant générer des programmes exécutables par des automates, des baies robotiques ou des commandes numériques. Certaines suites logicielles peuvent générer, avec des post-processeurs, des programmes robots exécutés par les calculateurs des baies de commande. D'autres déportent dans un calculateur temps réel la génération de trajectoire et envoient des consignes de positions périodiques aux contrôleurs de robots.

La problématique du rapprochement entre les approches d'automatisme et de robotique est apparue dans les années 2010 [Nicolson 2016], [Unnikrishnan 2017]. Le constat de départ était la convergence entre les composants matériels présents dans les baies de commande de robot et les composants d'usage général des fabricants d'automatismes. Le second bénéfice attendu était la limitation du nombre de suites logicielles à acheter, installer et maintenir. Les aspects qui distinguent ces deux approches sont des différences de structuration de la conception du logiciel machine, des approches de développement très différentes entre ingénieurs automaticiens et ingénieurs robotiques et des contraintes de respect des normes de sécurité.

Le développement des fonctions de sécurité est traité de façon différente entre les fabricants de machines et les fournisseurs de robots, qui doivent, par ailleurs répondre à des contextes normatifs différents. Il s'agit des normes *Sécurité des machines : ISO 13849*, décembre 2015 et *Robots et dispositifs robotiques — Exigences de sécurité pour les robots industriels : ISO 1028* publiée en janvier 2011 et qui est en cours de révision. Pour les fabricants de robots, la connaissance complète de l'architecture de la machine permet de réaliser des fonctions intégrant le modèle cinématique du robot pour le contrôle de la validité de l'espace de travail et éventuellement le modèle dynamique pour la recherche d'impacts avec la surveillance des couples moteurs. L'approche des concepteurs de machine est basée sur le contrôle des axes de mouvement pour lequel le standard de sécurité logicielle fournit plusieurs blocs fonctionnels de surveillance. L'intégration de fonctions de sécurité prenant en compte l'architecture du robot est alors à la charge du concepteur et il est difficile, dans un automate de sécurité dont la puissance de calcul est limitée et les langages de programmation ont des fonctionnalités limitées, de recréer des fonctions de surveillance portant sur la position ou la vitesse du point de centre outil.

La possibilité prévue par [Nicolson 2016] de piloter des architectures robotiques par des systèmes de contrôle-commande d'automatismes s'est effectivement réalisée et les principaux acteurs de ce domaine proposent des bibliothèques logicielles très élaborées pour piloter des architectures robotiques. Certaines baies de commande de robot sont composées intégralement de composants d'automatisme standard, à l'exception de la carte de sécurité.

Pour le pilotage des robots à partir d'un PLC, [Unnikrishnan 2017] distingue trois types de configurations selon la prise en charge des fonctions robotiques (cf. figure 3.2) :

— *Self Hosted Kinematics* : cinématique auto-hébergée. Le processeur du contrôleur de mouvement prend en charge la planification des mouvements et les transformations cinématiques. Ceci n'est possible que pour des architectures mécaniques simples.

— *Self Hosted Custom Kinematics* : mécanismes personnalisés auto-hébergés. Les architectures mécaniques spécialisées conçues par les fabricants de machines de série (OEM) peuvent également être contrôlées par leurs développeurs qui implémentent les relations cinématiques dans le PLC.

— *Remote Hosted Kinematics* : mécanisme hébergé à distance. Les principaux fournisseurs de solution d'automation prennent en charge des robots tiers à l'aide de protocoles de communication propriétaires conçus en collaboration entre le fournisseur de commandes de mouvement et le fournisseur de robots. La programmation logique et l'initiation des commandes cibles seront effectuées sur le PLC. Dans un tel système, le contrôleur de robot assurera à la fois la planification de trajectoire et les transformations cinématiques.

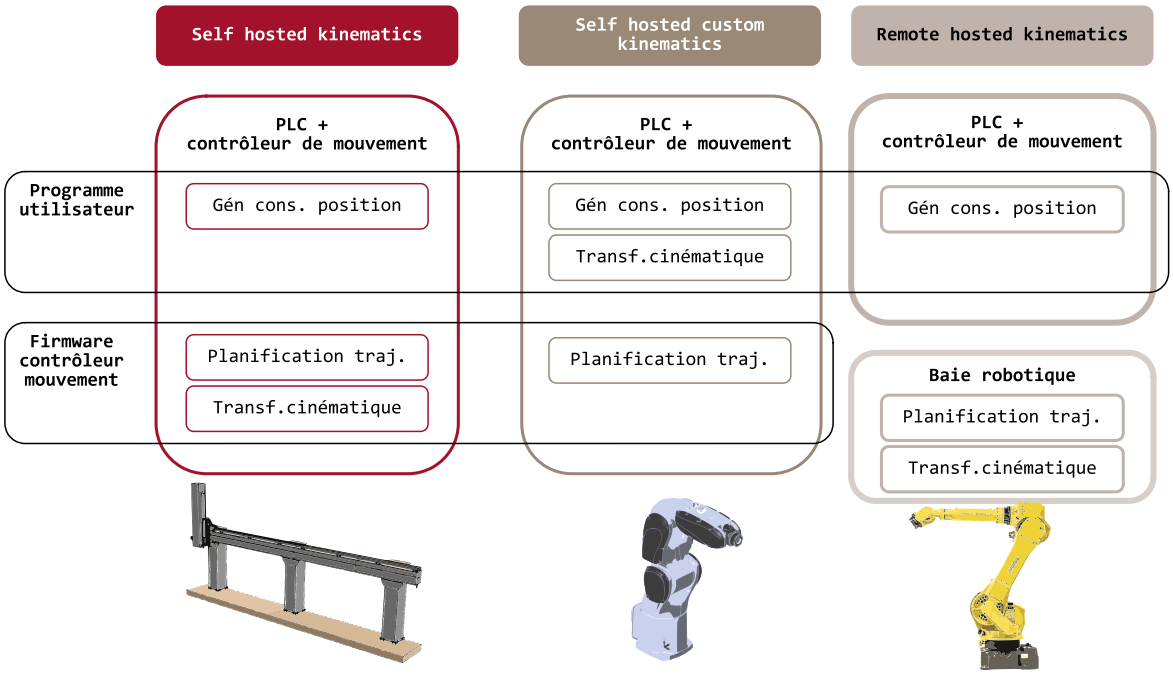


FIGURE 3.2 – Solution de contrôles d'architectures robotique depuis un PLC.

Cependant, il n'y a pas eu de convergence entre les solutions logicielles des fabricants d'automatismes et de robots. Avec l'apparition des cobots qui ajoutent d'autres modes de pilotage que le contrôle de position et la possibilité d'utiliser d'autres langages de programmation et de nouveaux types d'interfaces utilisateurs, l'offre de solutions techniques s'est encore étoffée.

Parmi les grands acteurs industriels, plusieurs stratégies sont actuellement en phase de déploiement. Une première est portée par le protocole de communication OCP-UA, présenté au

chapitre 2.1.2.2 qui cherche à proposer un modèle unifié de description et d'échange d'information avec les robots industriels des principaux acteurs du secteur. Un deuxième travail de standardisation est porté par la fondation PI qui a la charge de soutenir le développement des protocoles industriels PROFIBUS et PROFINET. En 2021, cette association a publié la version initiale de la spécification d'une bibliothèque de communication haut niveau entre robots industriels et PLC nommée *SRCI* pour *Standard Robot Command Interface*. A la suite de cette première étape, un travail de concertation a été engagé avec PLCopen pour envisager un travail d'unification du standard de commande multiaxe *PLCopen Motion Control Part 4 - Coordinated Motion*, avec la création d'un groupe de travail auquel j'ai pu participer.

3.1.1 Démarches de standardisation du pilotage multiaxe

3.1.1.1 Coordination de mouvement selon PLCopen

Dans l'approche PLCopen, le contrôle des machines multiaxes est traité de deux façons complémentaires. Il apparaît sous la forme de la création de relations de dépendance entre des axes esclaves et un axe d'entraînement principal (dans le document PLCopen Motion Control partie 1 et sous la forme de la coordination de mouvement entre axes articulaires pour obtenir le mouvement de l'effecteur (PLCopen Motion Control, partie 4).

Dans le cas de la conception de machines dédiées à la réalisation d'une tâche, il y a généralement un axe virtuel ou réel qui donne la cadence de fonctionnement de la machine ou de la ligne de production. Tous les mouvements complémentaires sont synchronisés sur celui de l'axe principal. Les dépendances entre axes sont de type boîte de vitesse, c'est à dire, qu'il y a une relation de proportionnalité entre le mouvement complémentaire et le mouvement principal avec éventuellement une possibilité d'avoir un décalage temporel pour ajouter une possibilité de réglage lors du fonctionnement ou pour corriger une dérive. Les situations de dépendance complexe entre le mouvement d'entraînement et un axe suiveur sont réalisées par des fonctions de cames électroniques qui décrivent les relations entre vitesses ou entre positions de de ces deux axes, généralement sous la forme de fonctions polynomiales. Dans le cas de la réalisation de machines généralistes dédiées à la génération de mouvements, la notion d'axe d'entraînement n'existe plus et c'est le suivi de consignes dans l'espace opérationnel qui devient le point d'entrée du calcul de l'évolution des positions des moteurs.

Le document de PLCopen *Function Blocks for motion control : Part 4 – Coordinated Motion* porte sur la commande de groupes d'axes générant des mouvements multidimensionnels. La version 1.0 de ce document a été publiée en 2008, la version 2.0 rédigée en 2021, est en toujours en cours de finalisation. L'approche retenue est une extension de la commande uniaxiale définie dans les parties 1 et 2 du standard PLCopen Motion Control, elle présente :

— la définition des systèmes de coordonnées utilisables et des relations de transformation cinématiques entre eux,
— un ensemble de blocs fonctionnels décrivant des mouvements obtenus en combinant les actions des différents axes,
— un diagramme d'états du groupe d'axes qui coordonne les diagrammes d'états des axes élémentaires.

Les systèmes de coordonnées définis dans le standard sont les suivants :

— *ACS* : Axis Coordinate System : positions articulaires liées aux axes élémentaires,
— *MCS* : Machine Coordinate System : positions dans un repère fixe lié à la machine,

— *PCS* : Product Coordinate System : repère lié à la tâche dans l'espace de travail de la machine, il est défini par une relation de transformation géométrique à partir du repère MCS qui peut évoluer au cours du temps (il peut être lié au déplacement d'un convoyeur, par exemple).

Les positions définies dans ces différents repères sont liées par des relations cinématiques (entre *ACS* et *MCS*) et par des transformations cartésiennes ou cylindriques entre *MCS* et *PCS*. Les blocs fonctionnels se répartissent en plusieurs catégories :

— les blocs d'administration qui permettent :
  — d'associer / de séparer les axes d'un groupe,
  — d'affecter un modèle cinématique au groupe, le format de définition du modèle robotique est laissé au choix du fournisseur de solution logicielle,
  — de définir la fonction de conversion entre repères de travail.
— et les blocs fonctionnels de mouvement qui sont de deux types :
  — des déplacements point à point, interpolés dans l'espace articulaire,
  — des déplacements définis dans l'espace opérationnel, avec une interpolation linéaire ou circulaire ou avec une solution générique de génération de trajectoire.

Des fonctions complémentaires permettent de coordonner le mouvement d'un groupe d'axes avec un mouvement complémentaire externe pour réaliser des tâches de suivi d'objet, de dépose de produit (colle, peinture) ou de prise au vol. Ces fonctions de synchronisation ou de tracking (suivi du déplacement d'un convoyeur ou d'une table rotative) créent une relation maître-esclave (comme cela peut être le cas pour les axes individuels) entre l'axe externe et le groupe d'axes, ou inversement. Il est également prévu de pouvoir coordonner deux groupes d'axes entre eux avec une relation de transformation géométrique modifiable entre eux.

Les blocs fonctionnels de mouvement ont des entrées qui permettent de définir les transitions entre différentes commandes (cf. figure 3.3). Ce sont des extensions aux groupes d'axes de l'entrée *BufferMode* des blocs fonctionnels de mouvement présentés dans la section 2.1.4.3.

L'entrée *BufferMode* décrit le type de succession entre les deux mouvements, elle peut prendre les valeurs suivantes :

— *Aborting* interruption du premier bloc de mouvement et activation immédiate du second,
— *Bufffered* le second mouvement commencera lorsque le premier sera terminé,
— *Blending-Previous / -Next / -High / -Low* la vitesse de transition est limitée à celle du premier ou du second mouvement, ou à la valeur maximale ou minimale des deux vitesses.

Les entrées *TransitionMode* et *TransitionParameter* décrivent la phase de raccord de la trajectoire au moment du changement de mouvement. Elles peuvent imposer la vitesse au moment du changement de consigne, une vitesse constante dans la phase de transition, la distance entre la trajectoire et l'intersection des deux segments (*P1P2* et *P2P3* sur la figure 3.3) ou d'autres contraintes laissées au choix des fournisseurs de solutions logicielles.

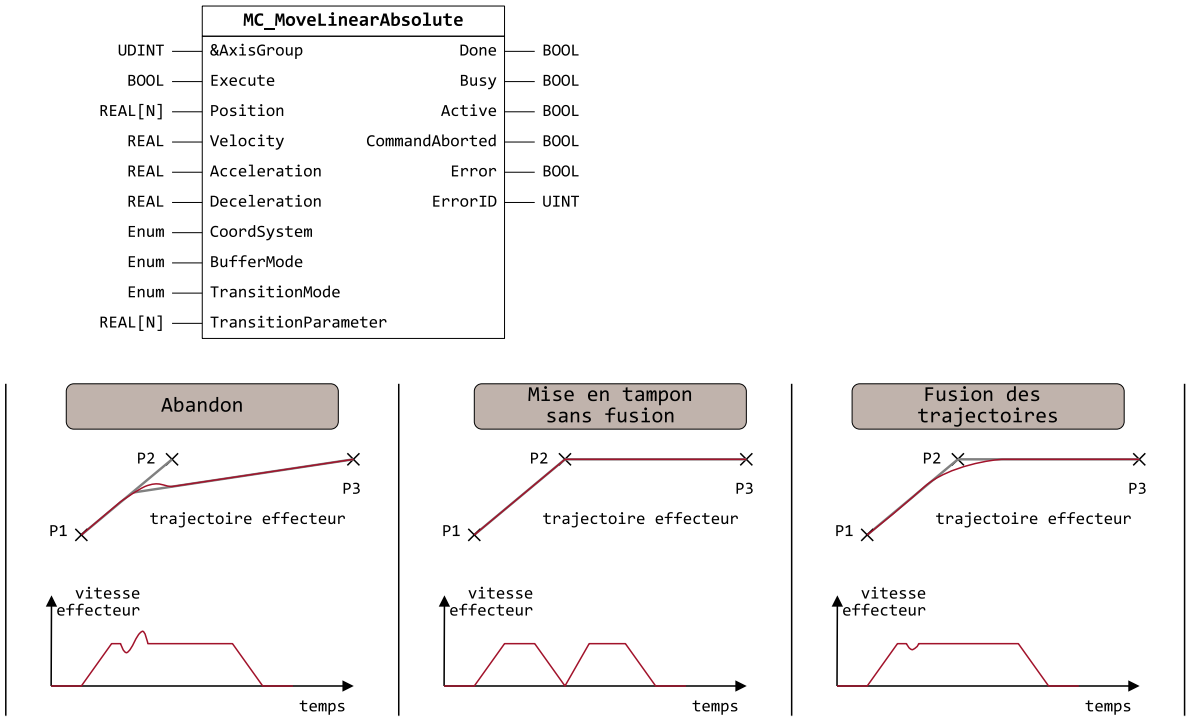


FIGURE 3.3 – Exemple de bloc fonctionnel de mouvement d'un groupe d'axes et illustration des types de transition entre deux déplacements *MC_MoveLinearAbsolute* vers les destinations P2 puis P3.

3.1.1.2 Exemple de fonctionnement d'une bibliothèque robotique propriétaire

La figure 3.1.1.2 présente la chaîne de traitement de la génération de mouvement de la solution logicielle *mappRobotics* de B&R Automation. Elle permet de prendre en compte des architectures robotiques standards (bras de robot sériel, SCARA, robots parallèles Delta) ou de définir des architectures spécifiques. La stratégie de génération de mouvement est identique quelle que soit l'architecture.

La description du comportement attendu du robot est programmée dans les langages du standard IEC61131-3 avec les blocs fonctionnels de mouvement PLCopen Motion Control, partie 4 ou dans un fichier de script qui sera lu et interprété à l'exécution du programme. Ce second mode est identique au fonctionnement des contrôleurs de robots industriels et permet de charger des trajectoires ou des séquences de fonctions robotiques sans recompiler le code du logiciel.

Les ordres de mouvement sont les entrées de l'ensemble de fonctions logicielles *Motion chain* qui réalisent :

— la transformation des blocs fonctionnels en séquence de commandes de mouvement et l'interprétation des scripts
— le calcul de la trajectoire à partir la séquence de commande de mouvement en cours d'interprétation,
— l'adaptation de la trajectoire en fonction des limites des axes,
— l'inversion du modèle cinématique,

— la génération de consignes articulaires périodiques envoyées aux contrôleurs de mouvement.

Le traitement des programmes est décomposé en consignes de mouvement élémentaires nommées *Motion packets* qui sont traités par les modules logiciels de la *Motion Chain* en tâche de fond, au fur et à mesure du déroulement des scripts.

La figure 3.5 présente un exemple d'exécution par le processeur des tâches de génération de mouvement pour la partie robotique. La partie « génération de consigne » est exécutée périodiquement dans la classe de tâches la plus rapide. Les fonctions de préparation de la trajectoire sont exécutées en arrière-plan, avec les services systèmes non prioritaires. L'exécution de ce traitement dépend de la charge de travail du processeur et peut conduire à l'interruption temporaire du mouvement du robot.

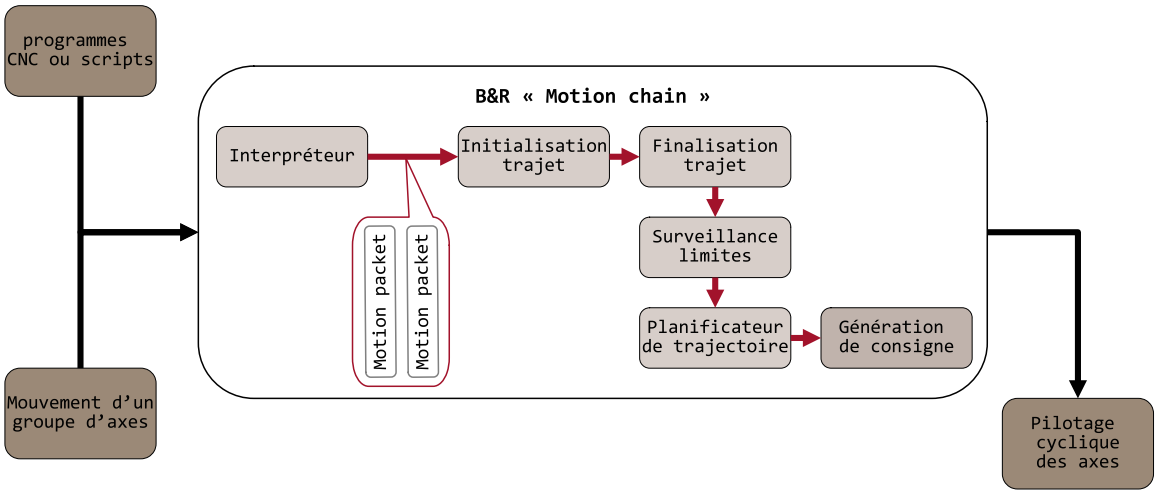


FIGURE 3.4 – Chaîne de traitement logiciel d'un programme de génération de mouvement robotique, par la solution mappRobotics de B&R Automation.

Le développement par les principaux fournisseurs de matériel d'automatisme de bibliothèques robotiques permet aux utilisateurs de réaliser des programmes de commande machine avec un haut niveau d'intégration d'architectures robotiques dans les situations où le PLC interagit avec les contrôleurs de mouvement ou avec une baie robotique donnant accès au contrôle en position articulaire. Ces bibliothèques fournissent également des fonctions complémentaires permettant :

— de charger des scripts qui permettent de modifier le fonctionnement du robot pendant l'exécution du programme automate,
— d'exécuter des programmes en code G, pour les utilisateurs de machines d'usinage, par exemple,
— de choisir les repères de travail et les repères outils,
— d'ajouter des fonctions de sécurité robotiques dans les programmes des automates de sécurité,
— de synchroniser simplement les mouvements du robot avec les mouvements d'axes externes.

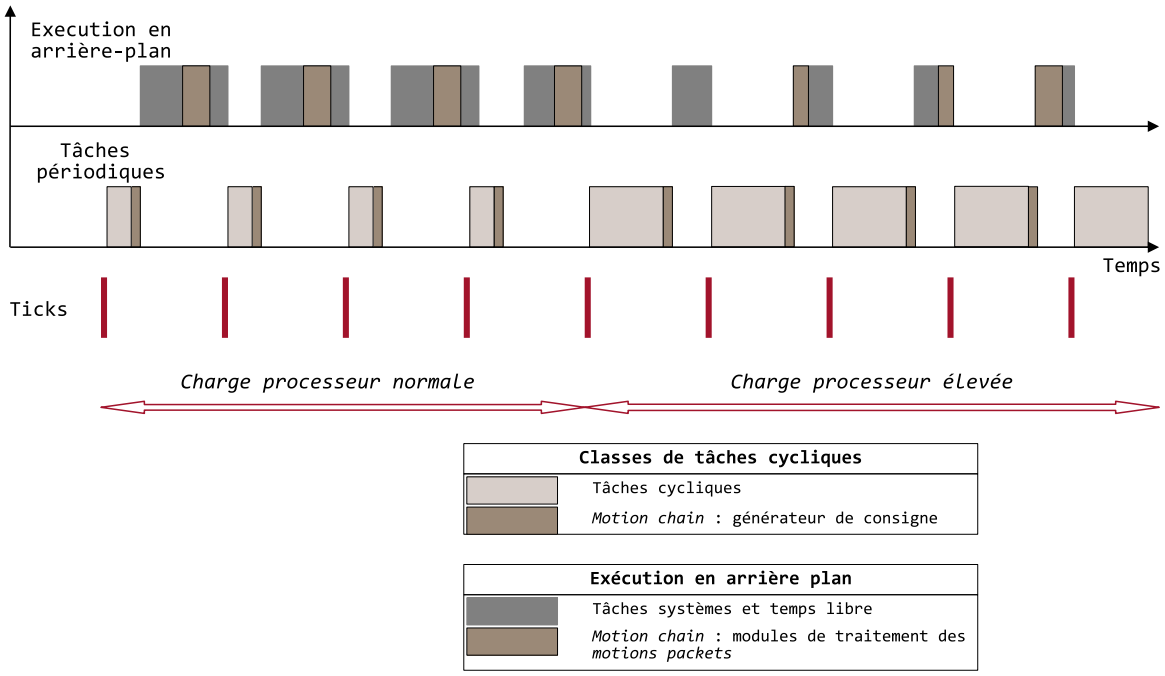


FIGURE 3.5 – Exécution périodique et en arrière plan des modules de la motion chain B&R Automation.

La solution B&R Automation permet également de définir des architectures mécaniques personnalisées en fournissant les modèles cinématiques tout en bénéficiant de la chaîne de génération de mouvements robotiques.

Cependant, ces solutions sont propriétaires et sont moins identiques entre fournisseurs de matériel que les bibliothèques de commande d'axe qui répondent à un standard plus contraignant, ce qui rend difficile de mutualiser des développements pour les mettre en œuvre sur des plateformes concurrentes. La stratégie de traitement des ordres de mouvement est par ailleurs très consommatrice de ressources et est peu adaptée au développement de systèmes réactifs par rapport aux évolutions de l'environnement.

Le chapitre suivant présente une autre stratégie de standardisation du contrôle des robots industriels par un calculateur externe.

3.1.1.3 Interface standard robot - PLC selon la fondation PI

L'entreprise Siemens et l'association PROFIBUS & PROFINET International (PI) ont entamé en 2021 la création d'une interface logicielle unique permettant de contrôler des baies de robots industriels à partir d'automates programmables en collaboration avec plusieurs fabricants de robots industriels. La dernière version de la librairie robotique SIMATIC de Siemens implémente ce standard qui permet de faire exécuter, par un programme automate, des fonctions des contrôleurs de robots. Un travail collaboratif entre PI, PLCopen et l'association allemande VDMA essaie de rendre compatibles les standards.

L'approche développée par *PI* s'appelle *Standard Robot Command Interface to PLCs* [AG 2021]. Elle porte sur la définition d'un mécanisme de communication et sur la spécification de deux bibliothèques, l'une destinée à être implémentée côté PLC, la seconde côté contrôleur de robot. La réalisation des bibliothèques est à la charge des fournisseurs de composants.

L'objectif de l'interface robotique est de pouvoir développer le logiciel de pilotage d'un robot à partir de blocs fonctionnels de la bibliothèque de l'automate, sans nécessiter de programmation supplémentaire du côté du contrôleur de robot. Il s'agit donc d'une topologie de type *Remote hosted kinematics* où l'automate génère la séquence d'ordres de mouvements et reçoit les informations d'état du robot. La prise en charge du modèle cinématique et la génération de trajectoire sont à la charge du contrôleur de robot.

La communication entre les deux contrôleurs repose sur un protocole de communication basé sur Ethernet, non déterministe. La solution Siemens est construite sur une communication *Profinet*, mais l'interface *SRCI* n'est pas dépendante de cette solution. Le PLC et le contrôleur de robot échangent des datagrammes qui contiennent les données nécessaires à la réalisation des ordres de mouvements (dans le sens PLC vers robot) et les informations d'états du robot dans l'autre sens. Les deux calculateurs travaillent à leurs périodes respectives et la communication entre les deux permet un échange cyclique d'informations avec une période moyenne plus élevée. Cette solution permet aux fournisseurs de matériel de développer leurs solutions de façon relativement indépendante mais pose de nombreuses difficultés pour le contrôle fin des trajectoires ou lors de changement d'objectif en cours de mouvement.

La spécification *SRCI* prend également en charge les modes de contrôle standardisés des robots industriels, adaptés à un pilotage externe (mode automatique et modes de test à vitesses contrôlées), l'intégration de certaines fonctions de sécurité logicielles et des fonctionnalités dédiées à la gestion des outillages. Elle crée une interface homogène avec des quasi-machines ayant des définitions géométriques et des principes de fonctionnement parfois différents (par exemple, le paramétrage des transformations entre repères est différent selon les fabricants de robots).

La figure 3.6 présente de façon simplifiée la décomposition fonctionnelle proposée par la spécification SRCI. L'utilisateur développe son programme de gestion machine dans l'automate. Le fournisseur d'automate livre une bibliothèque donnant accès aux fonctionnalités robotiques, et des services permettant de mettre en œuvre le protocole de communication. Les blocs fonctionnels du programme utilisateur sont convertis en instructions de commande (par exemple `MOVE_LINEAR`), particularisées par des arguments (`Position`, `Velocity`, ...) et qui attendent en réponse un retour d'état ou de réalisation de la part du logiciel robot (`ACTIVE`, distance restant à parcourir ...).

Le fabricant de robot fournit une solution embarquée dans le contrôleur de robot. Celle-ci :

— met en œuvre, la communication, côté serveur,

— et transforme les ordres de mouvement en tâches qui vont interagir avec le générateur de mouvement et les autres fonctions robotiques disponibles.

Ce partage de tâches entre deux calculateurs introduit un nombre important de services intermédiaires. Les messages qu'ils échangent sont transmis par un protocole non déterministe à période moyenne plus longue que les périodes de travail des calculateurs. Côté robot, le générateur de trajectoire a besoin de connaître plusieurs commandes de mouvement à l'avance pour gérer les transitions. Toutes ces contraintes pénalisent la réalisation du contrôle-commande global. Elles imposent de mettre en œuvre :

— une mise en mémoire tampon des commandes de mouvement avec un traitement diffé-

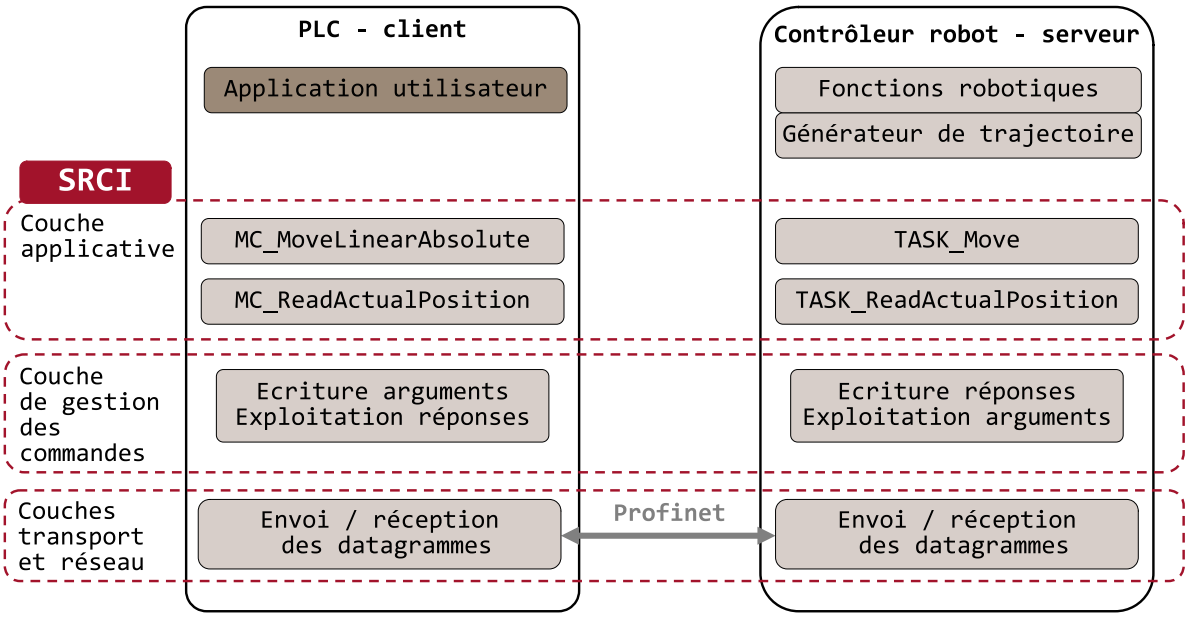


FIGURE 3.6 – Architecture fonctionnelle de la solution SCRI.

rencié suivant le mode d'exécution demandé (exécution des tâches en séquence, parallèle ou sur évènement)

— un mécanisme de déclenchement d'action (`trigger`) permettant de réagir rapidement à des évènements détectés par des capteurs externes (capteurs de présence, information caméra, capteur de position d'un convoyeur) câblés sur des cartes d'entrée de l'automate ou du robot.

Pour le développement de l'application côté automate, l'exécution distante des fonctions de mouvement a un coût important sur la garantie du comportement déterministe du logiciel de commande de la machine.

Par rapport aux blocs fonctionnels du standard monoaxe, les blocs fonctionnels de la solution SRCI présentent une interface plus complexe(cf. figure 3.7). Il est alors difficile d'interpréter les états de tous les blocs de mouvement pour connaître la situation instantanée du robot et la prendre en compte dans la couche logicielle de gestion machine.

La technologie SRCI est encore en cours de développement et un groupe de travail commun avec PLCopen a pour objectif de standardiser le développement des solutions de type *Remote Hosted Kinematics* où les fonctions robotiques sont appelées par le PLC de gestion d'une machine ou d'une ligne de production et où les fonctions robotiques sont exécutées dans les baies des robots. Les différences de paradigme de programmation, entre les scripts interprétés des contrôleurs de robot et les programmes exécutés périodiquement des contrôleurs industriels et l'exigence de développement une solution adaptée à tous les fabricants de robot rend difficile le travail de définition d'un standard.

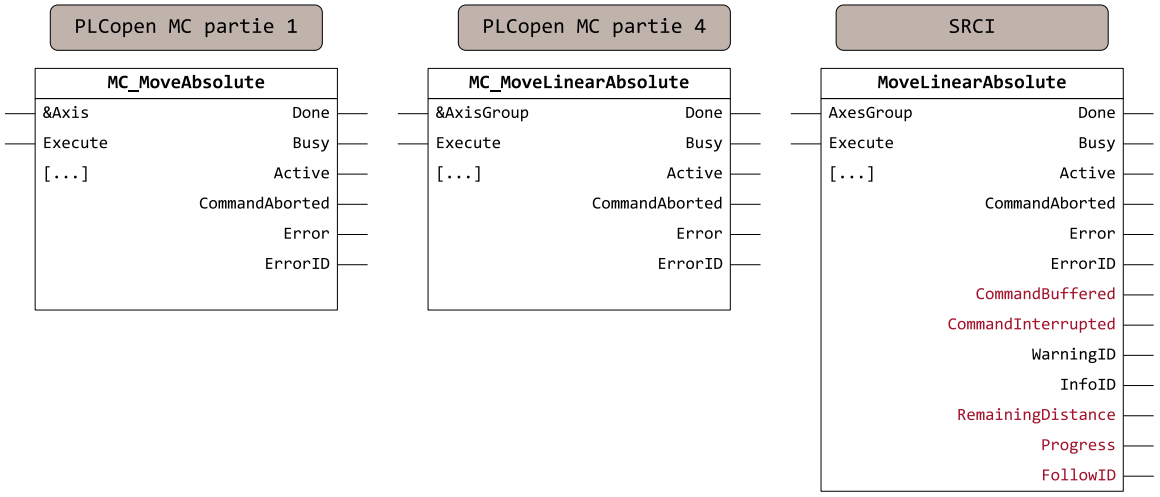


FIGURE 3.7 – Comparaison des sorties d'information entre les blocs fonctionnels des standards PLCopen Motion Control (partie 1 : axe, partie 4 : axes coordonnés) et SRCI.

3.1.2 Conclusion

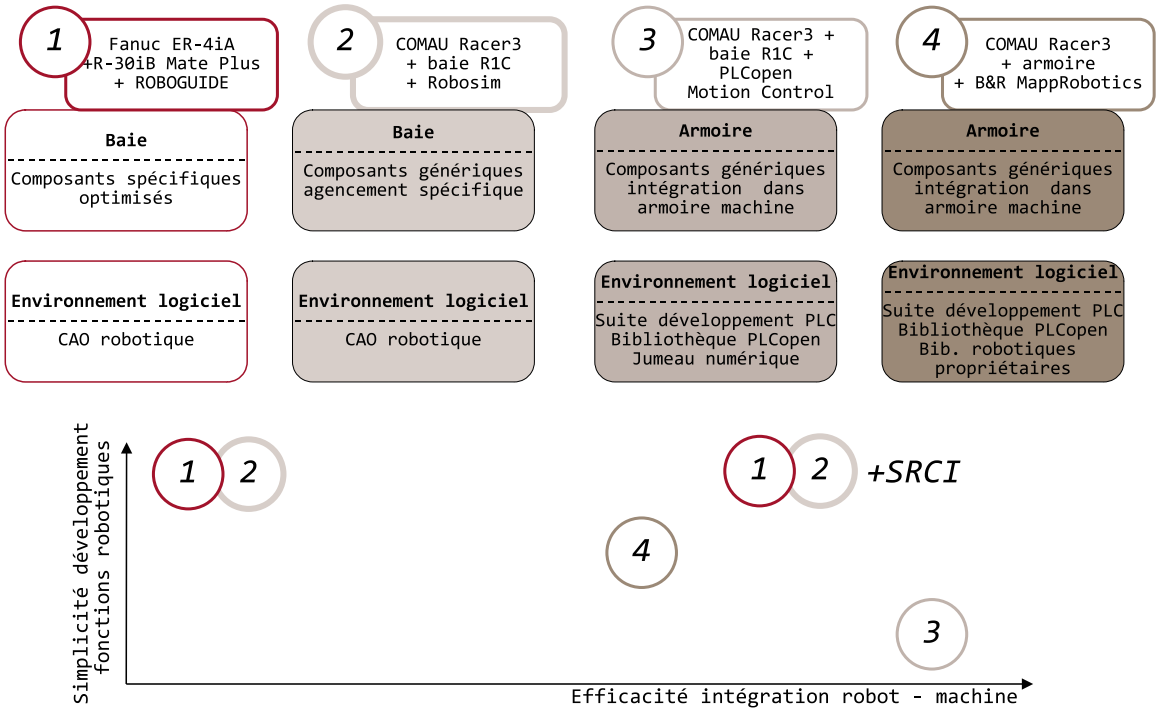


FIGURE 3.8 – Comparaison de différentes solutions industrielles de pilotages de robots.

L'offre de bibliothèques robotiques industrielles est très vaste et diversifiée. Pour les robots industriels standards, la situation actuelle où le fournisseur de matériel impose également l'usage de son logiciel de développement et son langage de programmation est en train d'évoluer. Les fabricants de matériel d'automatismes fournissent des bibliothèques robotiques propriétaires qui permettent de contrôler des architectures robotiques classiques ou personnalisables.

La figure 3.8 propose une classification des principales solutions technologiques selon deux axes :

— la possibilité de mettre en œuvre des fonctions robotiques évoluées (définition des trajectoires dans un outil de CAO, génération et simulation de scripts, identification des paramètres inertiels, optimisation de la dynamique du robot, gestion des outils, intégration de fonctions de sécurité liées au robot, intégration de capteurs de vision),
— l'aptitude à intégrer les axes d'un ou plusieurs robots industriels comme les autres d'axes de mouvement d'une machine de production.

Par rapport à ces deux critères, les solutions robotiques classiques, développées avec des composants spécifiques à partir de composants standards donnent accès à des fonctions robotiques évoluées mais nécessitent le développement d'un logiciel spécifique côté robot et une coordination entre les calculateurs. L'exemple 4 correspond à une solution propriétaire d'un fournisseur de matériel d'automatismes, qui permet de développer l'intégralité du logiciel de commande dans l'environnement de développement automate et de l'exécuter sur un seul calculateur. Les architectures pilotables sont limitées aux contrôleurs robotiques ouverts ou à la fourniture d'une mécanique de robot nue (sans contrôleurs de mouvements). La génération de trajectoires à partir de scripts en langage interprété ou en code G offre la possibilité de changer les trajectoires du robot à l'exécution du programme. Cependant la présence d'un mécanisme de planification de trajectoire fonctionnant en tâche de fond avec un horizon de quelques ordres de mouvement limite les possibilités de prise en compte des évolutions dynamiques de l'environnement. L'adoption de la technologie SRCI déplace les solutions robotiques dans le sens d'une meilleure intégration des robots dans le logiciel machine et supprime la nécessité de développer le logiciel robotique dans un autre environnement. Cependant, la prise en compte des évolutions de l'environnement est encore plus difficile dans la mesure où la coordination des fonctions de mouvement et leur exécution sont réalisés par deux calculateurs distincts, sans exigence de communication périodique, ni de fonctionnement déterministe.

La solution 3, basée sur une architecture matérielle à base de composants standards, d'un logiciel intégralement exécuté par un calculateur industriel de type PLC et d'une bibliothèque robotique construite sur la couche *contrôle d'axe* du standard PLCopen Motion Control correspond à la démarche que nous avons mise en œuvre et qui sera détaillée dans la suite de ce manuscrit. Elle va nous permettre de dissocier la génération de trajectoire et le calcul des consignes articulaires pour implémenter, sur des architectures robotiques classiques ou non conventionnelles des algorithmes de contrôle basés sur la résolution de problèmes d'optimisation.

Le revers principal de cette approche est que les fonctions optimisées des robots industriels ne sont pas accessibles. Ceci limite les performances atteignables, par exemple en termes de précision de suivi de trajectoire, puisque les fonctions de correction de la dynamique du robot sont maîtrisées (et non partagées) par les fournisseurs de matériel. Il faut également remarquer que fournir toutes les fonctions disponibles nativement sur un robot industriel nécessiterait beaucoup de développement. Cependant, le principe d'utiliser un calculateur industriel permet également de choisir d'exploiter les solutions technologiques des fournisseurs de matériel d'automatismes,

si celles-ci s'avèrent être mieux adaptées au cahier des charges de la machine.

3.2 Proposition d'une architecture logicielle pour le contrôle de mécanismes

Le contexte d'usage de notre bibliothèque de composants de contrôle-commande est celui de systèmes polyarticulés, constitués d'une partie opérative présentant un ensemble de pièces mécaniques supposées rigides reliées entre elles par des liaisons. Nous allons distinguer deux parties dans l'ensemble de ces composants mécaniques. Nous nommerons *mécanisme* le système de transformation de mouvement qui relie les sorties des actionneurs aux mouvements dans les liaisons entre segments. La partie *robot* décrit les relations qui relient les mouvements entre segments et le déplacement des effecteurs, composants portés par les segments terminaux (parfois également par des segments intermédiaires) qui vont interagir avec la matière d'œuvre sur laquelle travaille le système automatisé. Dans le cas d'un système destiné à déplacer un objet, l'effecteur est un préhenseur, mais il peut également s'agir d'un outil, d'une buse de peinture ou de colle, d'une torche ou d'une broche.

La fonction globale d'un système multiaxe est de contrôler la position de son effecteur ou le torseur d'actions transmises par l'effecteur sur l'environnement à partir du pilotage des actionneurs. La séparation entre *mécanisme* et *robot* fait apparaître deux groupes de relations :

— des relations de couplage qui relient les positions et les actions mécaniques motrices en sortie des actionneurs et les positions et actions transmises dans les liaisons entre segments,

— des relations de changement d'espace entre les mouvements des articulations et celui de l'effecteur ou entre les actions mécaniques articulaires et transmises vers l'extérieur.

Cette distinction entre « mécanisme » et « architecture robotique » est une simplification de la représentation de systèmes multicorps. Elle est basée sur une distinction des fonctions réalisées par les composants de transmission de puissance d'une part, et par l'agencement des liaisons du système polyarticulé, d'autre part. Elle s'appuie également sur l'existence de gammes de problèmes différents : certains systèmes sont intégralement des mécanismes : il n'y a pas de chaîne de transformation de mouvement après le niveau articulaire. Dans ce cadre, les enjeux de la modélisation sont la représentation des couplages entre actionneurs et degrés de libertés pilotés et la possibilité de prendre en compte des modèles de transmission plus complexes, lorsque ce niveau de définition est nécessaire. En effet, si l'on souhaite, dans un objectif de contrôle ou de surveillance, exploiter les mesures de couple actionneur, il faut pouvoir prendre en compte les écarts de comportement des transmissions par rapport au modèle linéaire idéal utilisé en première approche. Lorsque ces grandeurs sont identifiées, un modèle de transmission plus précis peut intégrer la déformation de la chaîne de transmission et les pertes par frottement.

La partie « architecture robotique » correspond également à un ensemble de problèmes spécifiques, traité abondamment par la littérature scientifique, qui portent sur la conversion de mouvements ou d'efforts entre les espaces articulaires et opérationnels.

3.2.1 Implémentation d'une solution robotique générique

Notre cahier des charges du logiciel de contrôle-commande d'un système multiaxe présente la fonction centrale suivante : fournir à l'utilisateur (opérateur ou système de contrôle de plus

haut niveau) un ensemble de fonctions permettant de contrôler les mouvements du robot avec une interface unifiée.

Cette fonction peut se décomposer en un ensemble de fonctions techniques (cf. 3.9). Pour les applications que nous souhaitons mettre en œuvre, nous avons choisi d'implémenter dans l'architecture générique de contrôleur de robot un nombre limité de fonctions élémentaires :

— déplacer chaque articulation en mode manuel,
— déplacer l'effecteur vers une position d'arrêt par une interpolation dans l'espace articulaire,
— déplacer l'effecteur vers une position d'arrêt par une interpolation dans l'espace opérationnel,
— suivre des consignes périodiques de positions articulaires,
— suivre des consignes périodiques de la vitesse de l'effecteur,
— suivre des consignes d'efforts articulaires (sur les architectures qui le permettent, en cours de développement),
— fournir les informations d'état du système (états du robot et de ses composants, informations géométriques, cinématiques et énergétiques).

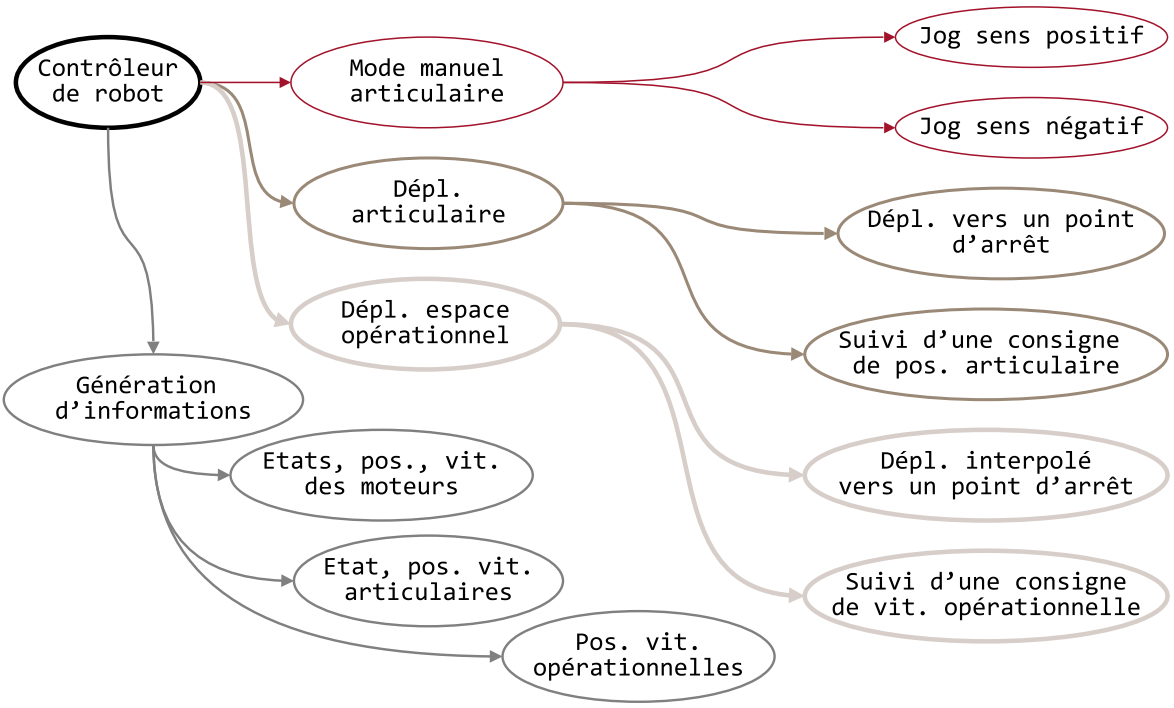


FIGURE 3.9 – Gestionnaire de robot : fonctions techniques implémentées.

A partir de l'identification de ces fonctions et des contraintes d'implémentation liées au type de matériel que nous avons choisi d'utiliser, il est possible de définir l'architecture de la bibliothèque de pilotage de robots. Pour cela, il est nécessaire de :

— définir les espaces de travail de robot,
— insérer une couche robotique dans la décomposition du logiciel de contrôle-commande,

entre la gestion des actionneurs et celle de la machine,
— définir les structures d'interface avec le programme utilisateur,
— ajouter un modèle abstrait de robotique dans la bibliothèque `rtrmac`,
— définir la stratégie de réalisation des fonctions complexes et identifier les bibliothèques externes à intégrer.

3.2.2 Décomposition hiérarchique du contrôle-commande

La figure 3.10 présente la décomposition de l'architecture des logiciels de contrôle-commande que nous avons réalisée. Cette décomposition est hiérarchisée par strates, de la couche proche du matériel à la couche supérieure de gestion machine qui sera elle-même intégrée dans la gestion d'un groupe de machines ou d'une ligne de production. Cette séparation en différentes couches prépare la répartition des tâches entre calculateurs et entre classes de tâches, selon le mécanisme présenté chapitre 2.1.3. Les couches inférieures ont besoin d'avoir une priorité plus importante et une période d'exécution plus courte que les couches suivantes.

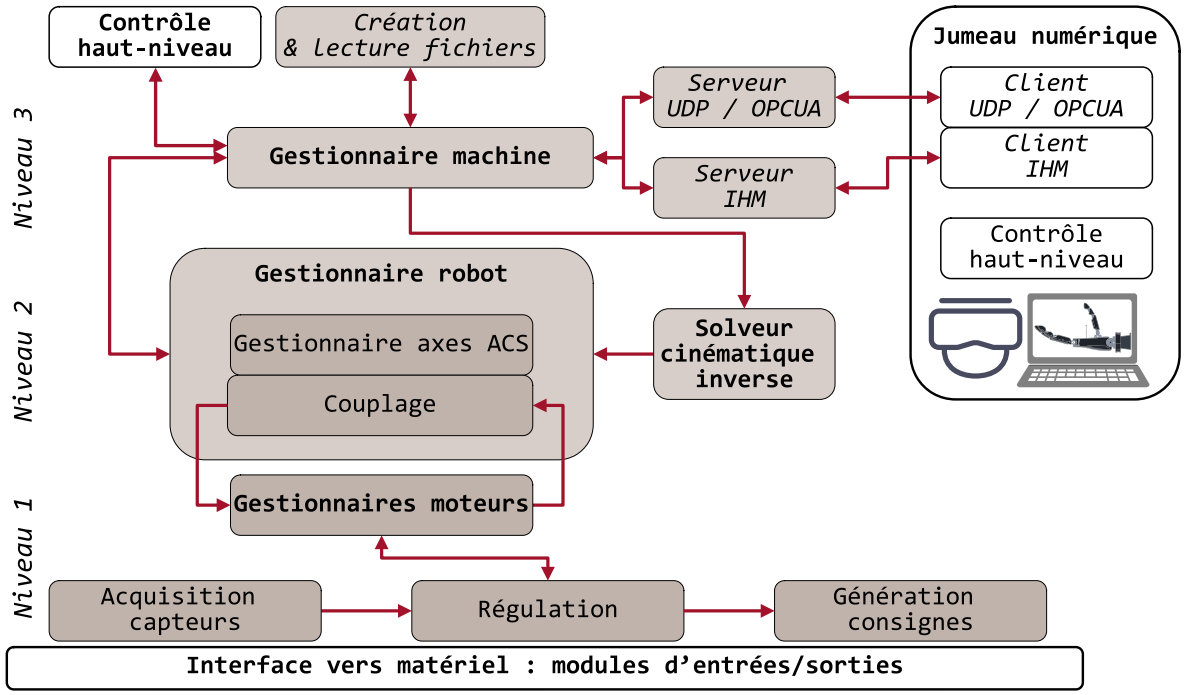


FIGURE 3.10 – Architecture générique du logiciel de contrôle robotique.

Le premier niveau réalise la conversion et le traitement des informations issues des capteurs, met en place les boucles de régulation correspondant à la topologie du système d'entraînement retenu (cf. chapitre 2.2.2) et met en forme les consignes pour les pré-actionneurs.

Le deuxième niveau exécute les gestionnaires d'axe moteurs. Le programme instancie et exécute les objets logiciels programmables (classes de contrôle-commande et classes de contrôle des paramètres) qui coordonnent l'usage des blocs fonctionnels PLCopen Motion Control.

Le troisième niveau est le gestionnaire de mécanisme ou de robot. Il exécute en interne des gestionnaires d'axes robotiques qui donnent accès au pilotage au niveau articulaire et masquent à l'utilisateur le pilotage direct des moteurs.

Le programme de gestion machine traduit la spécification d'une application particulière. Il sélectionne les fonctions robotiques utiles aux tâches à réaliser et les coordonne selon les instructions provenant de l'interface utilisateur. Si ces tâches sont définies dans l'espace opérationnel, la génération de mouvement articulaire est réalisée par un solveur externe au gestionnaire robot qui calcule les consignes de position par résolution d'un problème d'optimisation.

Nos logiciels de contrôle-commande sont associés à un jumeau numérique qui permet de mettre à disposition l'interface utilisateur et ajoute une représentation de la machine en environnement de réalité virtuelle. Cet aspect sera plus longuement détaillé dans le chapitre 4.1. Des algorithmes complémentaires permettant le contrôle haut-niveau du robot sont nécessaires. Ils peuvent être exécutés dans le contrôleur du robot ou à distance, par le jumeau numérique, par exemple.

Enfin, des tâches de service permettent d'échanger des informations sur un support Ethernet (par communication UDP ou OPC-UA) et de stocker des données calculées dans l'automate ou de suivre des séries de consignes fournies dans des fichiers.

3.2.3 Intégration dans la bibliothèque `rtrmac`

L'ajout de la fonctionnalité de pilotage de robots dans la bibliothèque *rtrmac* est réalisé par l'ajout d'un nouveau package, `RbRobot` qui contient la définition des nouveaux types de données, des nouvelles constantes et les classes permettant de définir deux unités d'organisation de programme :

- `RbMechanism` et `RbMechanismParamAuditor` sont des classes abstraites qui permettent le pilotage d'un groupe d'axes articulaires et la représentation des relations de couplage entre l'espace des moteurs et l'espace des coordonnées articulaires
- `RbRobot` et `RbRobotParamAuditor` sont des classes abstraites qui permettent le pilotage d'un groupe d'axes articulaires, la représentation des relations de couplage et les conversions entre espace opérationnel et espace articulaire.

Pour un nouveau robot ou un nouveau mécanisme, il faut créer :

- une bibliothèque C++ contenant les classes de contrôle-commande et de contrôle des paramètres qui dérivent d'un des deux jeux de classes de bases définies précédemment,
- et une bibliothèque IEC contenant les informations partagées avec les tâches codées avec d'autres langages : constantes définissant la taille des tableaux, définition du type de la structure de paramètres.

3.2.4 Définition des espaces de travail

La modélisation d'un système multiaxe en vue de son pilotage fait apparaître plusieurs espaces de travail qui vont permettre de décrire l'état du système ou de définir les consignes de génération de mouvement. Ce sont :

- L'espace des axes moteurs, repéré `Mot` qui donne accès au contrôle bas niveau des actionneurs via l'interface unifiée d'axe PLCopen Motion Control (cf. section 2.2.2.3).
- Axes articulaires pilotés : `Acs` (coordonnées exprimées dans l'espace articulaire, nommé « Axis Coordinate System » dans PLCopen Motion Control)

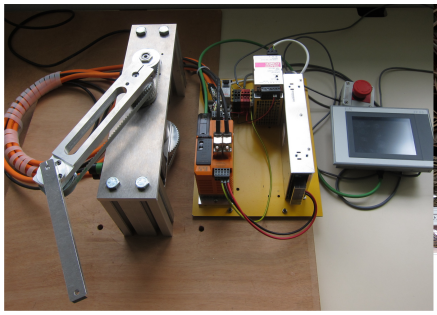
— Axes articulaires (axes pilotés et axes entraînés) : `Jnt` (Joints)
— Mouvements contrôlés de l'espace opérationnel : `Mcs` (coordonnées exprimées dans le repère machine, nommé « Machine Coordinate System »).

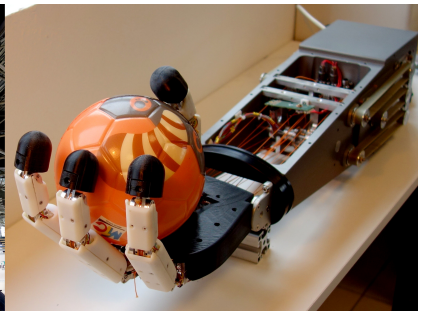
L'ensemble des mécanismes que nous souhaitons contrôler présente une variété qui nécessite de distinguer tous ces espaces. Certaines architectures sont en situation de redondance, le nombre de degrés de liberté actionnés est supérieur au nombre de dimensions contrôlées dans l'espace opérationnel. C'est le cas d'un bras robotique sur un axe linéaire : la cellule dispose de sept axes pilotables pour le placement en position et en orientation d'un effecteur. De façon opposée, certains mécanismes présentent un sous-actionnement : le nombre d'axes articulaires est supérieur au nombre d'actionneurs. Certains mouvements articulaires ne peuvent pas être directement contrôlés et dépendent des mouvements d'autres articulations.

Le tableau 3.1 présente plusieurs architectures pilotées dans le cadre de ce travail. Le robot 2R sériel représente la situation classique où les tailles de tous les espaces sont identiques, la cellule bras industriel et axe linéaire présente une redondance d'ordre 1. La main SeaHand est, quant à elle, fortement sous-actionnée.

Table 3.1 – Tailles des différents espaces pour plusieurs architectures de robots.

Système	Actionneurs	Articulations pilotées	Articulations	Espace opérationnel
Robot 2R enseignement	2	2	2	2
Bras 6 axes et axe linéaire	7	7	7	6
Main SeaHand	6	6	12	6



Pour chaque robot, il est nécessaire de définir les relations qui permettent de passer entre les différents espaces (cf. Figure 3.11). Les relations `MotToAcs` et `AcsToJnt` correspondent au couplage cinématique direct réalisé par le mécanisme de transformation de mouvement. Les relations `JntToAcs` et`AcsToMot` représentent les relations inverses. La relation `JntToMcs` calcule la position et l'orientation du ou des effecteurs en fonction des positions articulaires. La relation inverse est difficile à généraliser. Elle n'est pas nécessaire pour les fonctions que nous avons décrites précédemment et n'a pas vocation à être dans le contrôle-commande bas niveau : elle pourra être fournie par le jumeau numérique.

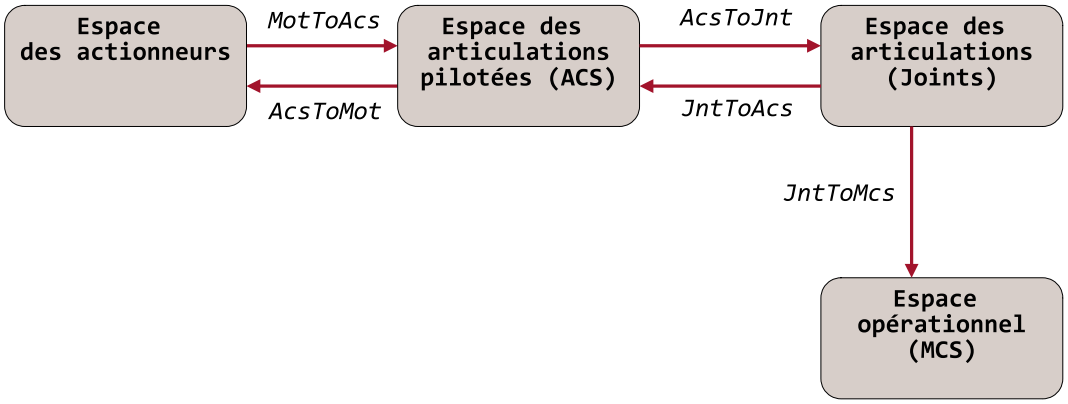


FIGURE 3.11 – Relations entre espaces de représentation du robot.

3.2.5 Intégration de bibliothèques tierces

Pour respecter notre objectif d'une dépendance minimale aux fabricants de matériel et aux environnements logiciels, nous avons choisi de limiter au maximum le nombre de bibliothèques tierces à intégrer dans les projets logiciels. Lorsqu'une dépendance devient nécessaire, il est important de retenir une solution stable, éprouvée en usage d'informatique temps-réel ou embarquée, si possible elle-même indépendante d'autres bibliothèques externes et dont les licences logicielles permettent l'usage en recherche et le transfert vers une solution industrielle.

Comme le domaine d'application des logiciels développés dans le cadre de ce travail n'inclut pas les fonctions de planification ou d'intégration de vision fournies par les intergiciels robotiques, les besoins d'intégration que nous avons rencontrés pour l'instant sont peu nombreux et nous n'avons ajouté que trois dépendances. Ces bibliothèques sont utilisées dans les programmes en C++. La démarche de création d'une interface pour utiliser une fonction externe dans les programmes en langages automates a été présentée au chapitre 2.3.2.

Eigen Les langages IEC 61131-3 et le C++ ne disposant pas d'une bibliothèque permettant de manipuler les matrices, nous avons choisi d'intégrer *Eigen* pour les opérations matricielles et les manipulations géométriques.

Eigen est une bibliothèque d'analyse numérique en C++ développée par Inria pour fournir des outils d'algèbre linéaire, les opérations matricielles et vectorielles, les transformations géométriques et des solveurs numériques. C'est un logiciel libre sous licence Mozilla Public License 2.0 qui a une communauté d'utilisateurs importante. Elle est intégralement composée de fichiers d'en-têtes ce qui la rend très facile à intégrer et applique le principe de la programmation générique avec les templates C++. Les matrices et vecteurs peuvent être alloués dynamiquement ou de façon statique, option que nous allons privilégier pour exécuter des programmes sur un runtime temps-réel.

Reflexxes Nous avons également besoin d'un mécanisme d'interpolation pour réaliser de la génération de mouvement simple, mais, contrairement aux générateurs de trajectoire des robots industriels, il est nécessaire que l'interpolation s'effectue de façon dynamique, à chaque pas de temps.

La bibliothèque de génération en-ligne de mouvements `Reflexxes` a été brièvement présentée au chapitre 2.3.2. Elle a été développée par l'entreprise Reflexxes, start-up essaimée du Robotics Institute de Technische Universität Braunschweig et qui a été rachetée par Google Inc.

Nous utilisons la version `II` de la bibliothèque `Reflexxes` qui a été publiée sous Licence publique générale limitée GNU, ou GNU LGPL (pour GNU Lesser General Public License). Cette bibliothèque permet de générer en-ligne des lois de mouvement à vitesses continues avec des bornes sur les vitesses et les accélérations. La destination des mouvements et ces bornes peuvent évoluer en cours de déplacement. `Reflexxes type IV`, sous licence commerciale, génère des mouvements pour lesquels les accélérations sont continues et les surraccélérations sont bornées.

qpOASES Pour développer des systèmes de contrôle réactif, nous avons besoin de résoudre des problèmes d'optimisation sous contraintes : la génération de consigne de vitesse doit produire des mouvements réalisables par le mécanisme, à chaque instant.

qpOASES [Ferreau 2017] est une bibliothèque open source de programmation quadratique réalisée à partir des travaux de H.J. Ferreau en 2006 pour réaliser un contrôle embarqué de la combustion d'un moteur diesel [Ferreau 2007], [Ferreau 2014]. Elle est publiée sous licence GNU Lesser General Public License (LGPL). Elle met en œuvre une stratégie de résolution rapide d'un problème d'optimisation sous contraintes qui est adaptée à un usage périodique pour un problème dont la formulation peut changer à chaque pas de temps. Les performances de qpOASES vont, dans un premier temps, permettre de résoudre l'inversion du modèle cinématique d'un robot sur les cibles que nous utilisons, avec une période de génération de consigne de l'ordre de 2ms. Le calcul des vitesses articulaires pourra, dans un second, répondre à des problèmes d'optimisation plus complexes.

3.3 Pilotage des mécanismes

Le premier type de système multiaxe que l'extension de la bibliothèque `rtrmac` permet de contrôler répond à la définition de mécanisme. Il s'agit d'un système réalisant une transformation de mouvement entre un ensemble d'actionneurs (dans notre cas, électriques : moteurs ou moteurs linéaires) et un ensemble d'axes de mouvement qui vont directement remplir des fonctionnalités.

La figure 3.12 présente les classes et la structure qui construisent l'unité d'organisation de programme de contrôle des mécanismes. Les classes `RbMC_Mechanism` et `RbMC_MechParamAuditor` sont des classes abstraites qui dérivent des classes `RbLConCParam` (qui implémente le modèle *Level-controlled* sans état d'achèvement) et `RbParam`, respectivement. Elles seront elles-mêmes dérivées pour décrire le comportement de mécanismes spécifiques et seront associées à une nouvelle structure de paramètres qui aura, au moins les champs présents dans la structure de base `MechParam_typ`.

L'interface des classes dérivant de `RbMC_Mechanism` est la suivante, à chaque cycle :

— un tableau de structures d'informations d'axes robotiques renseigne sur l'état courant des moteurs (variable `mInfo`),

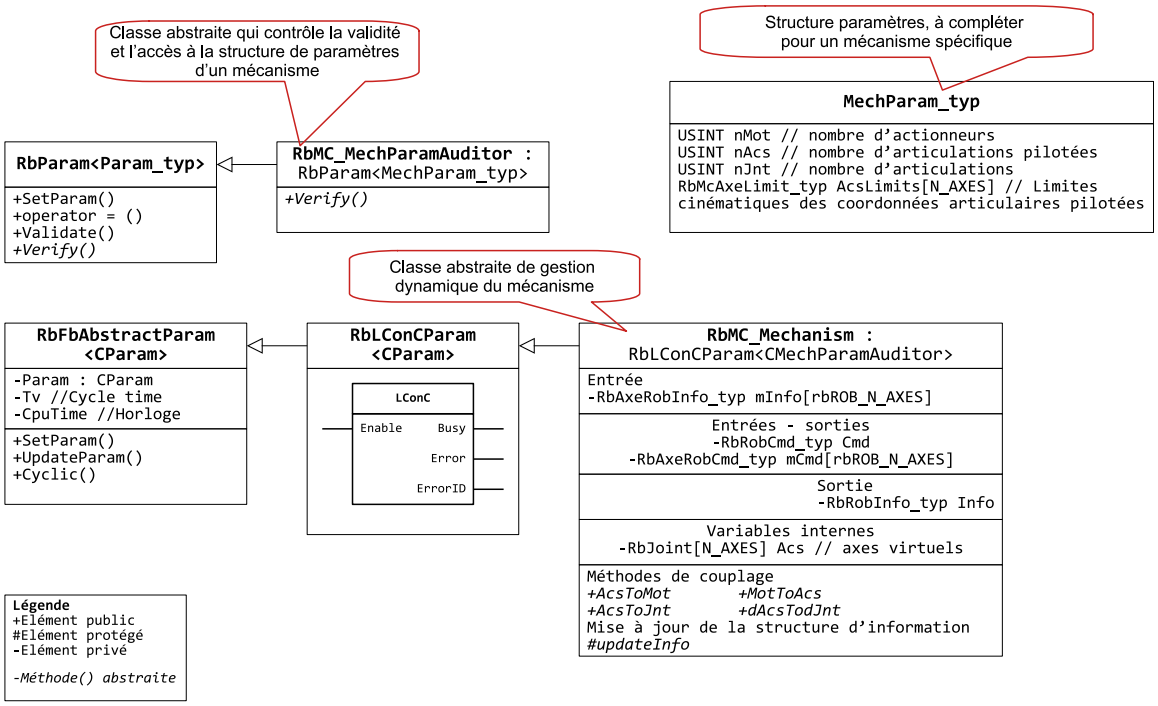


FIGURE 3.12 – Classes de base des modèles de mécanismes dans `rtrmac`.

— un tableau de structures de commandes des axes moteurs est lu et peut être modifié (variable `mCmd`, tableau de structures de type `RbAxeRobInfo_typ`, cf. chapitre 2.2.2.3),
— la variable `Cmd` contient les ordres envoyés par le gestionnaire machine au gestionnaire de mécanisme (variable `mCmd`, tableau de structures de type `RbAxeCmd_typ`),
— la sortie `Info` permet de renseigner le gestionnaire machine sur l'état de tous les axes et du mécanisme.

La figure 3.13 décrit les échanges d'informations entre programmes. Le programme qui exécute les gestionnaires d'axes robotiques offre une interface unifiée vers les pré-actionneurs. Le programme de gestion de mécanisme ou de robot modifie les structures de commande des axes robotiques et utilise les structures d'information pour actualiser sa propre structure d'information et pour faire évoluer sa machine d'états. Ce programme échange des informations avec le programme de gestion machine qui a un rôle de coordination et d'interface avec l'utilisateur. Le programme de gestion modifie la structure de commande `Cmd` du robot pour déclencher des actions et lit l'état du robot dans la structure d'information `Info`.

L'unité d'organisation de programme `mécanisme` remplit les fonctions suivantes :

— exécution des gestionnaires d'axes articulaires sous la forme d'axes virtuels de type `RbJoint`,
— administration du groupe d'axes articulaires,
— administration du groupe d'axes moteurs externe,
— établissement d'une relation de couplage entre moteurs et articulations, et entre articulations pilotées et articulations liées,

— coordination des prises d'origine entre les deux groupes d'axes,
— mise à disposition de mouvements dans l'espace articulaire.

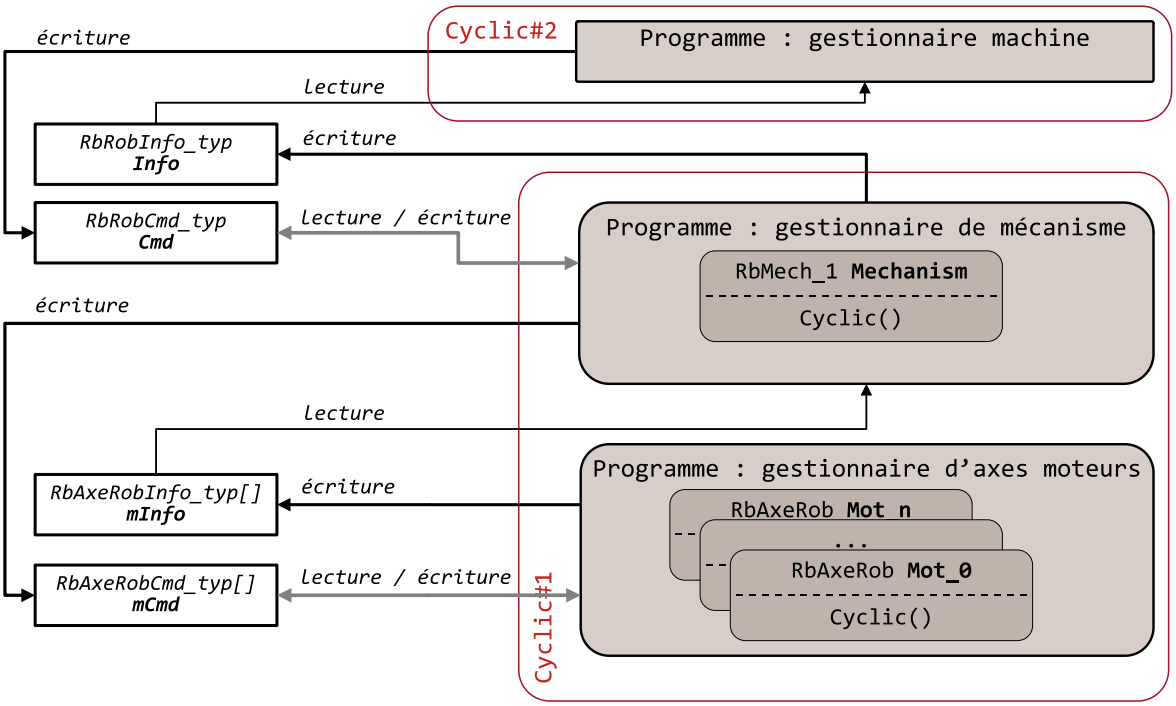


FIGURE 3.13 – Échange d'information entre programmes : gestionnaires d'axes moteurs, de mécanisme et machine.

Les deux sections suivantes présentent plus précisément la démarche d'association des axes articulaires et des axes moteurs, puis les modes de pilotage possibles.

3.3.1 Description du couplage moteurs - articulations

Notre démarche de conception d'un gestionnaire de mécanisme repose sur les principes suivants :

— utiliser comme composants de base les unités d'organisation de programme d'axes robotiques présentées précédemment (cf. chapitre 2.2.2),
— externaliser les gestionnaires d'axes moteurs, ce qui permet de les tester et de régler leurs paramètres indépendamment du fonctionnement en groupe d'axes,
— réaliser un couplage entre axes en passant les moteurs en suivi de consigne périodique, de position ou de couple.

La description des phases de préparation du mécanisme (démarrage, arrêt, prise d'origine, création et rupture de la relation de couplage entre les axes moteurs et les axes articulaires) et la gestion de l'arrêt d'urgence logiciel sont présentées dans l'annexe A.

La difficulté de réalisation d'un gestionnaire de mécanisme générique repose en partie sur la diversité des situations de prises d'origine, cet aspect est également discuté dans l'annexe A.

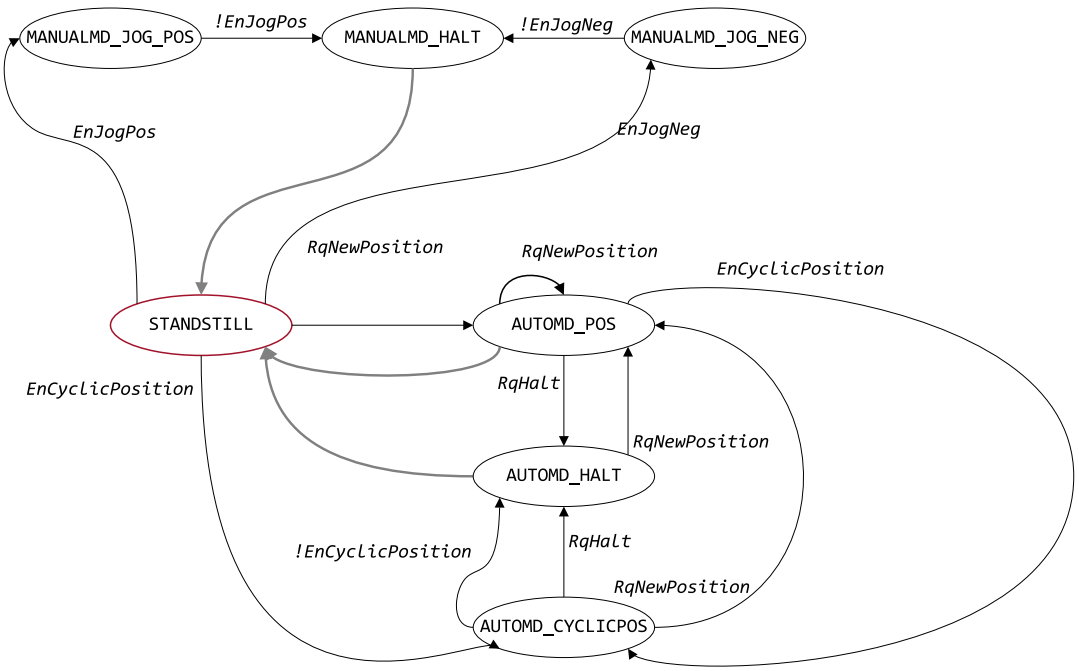


FIGURE 3.14 – Classes mécanisme et robot : activation des modes de contrôle en position.

La figure 3.14 présente les modes de déplacement en position articulaire utilisables et les transitions possibles entre ces modes :

— un mode de déplacement manuel de type *jogging* (`MANUALMD_JOG_xxx`), c'est un déplacement sur un seul axe tant qu'une demande d'avance ou de recul est active. Le mouvement s'arrête sur des positions de fin de course réglables.
— un déplacement vers une position d'arrêt avec coordination des mouvements ou non, avec une loi de vitesse saturée en vitesse et en accélération (`AUTOMD_POS`). Ce déplacement de base peut être interrompu par une demande d'arrêt avant d'atteindre l'objectif, ou par une nouvelle demande de mouvement avec un objectif ou des vitesses ou accélérations admissibles différentes. Ce fonctionnement correspond au mode d'interruption sans buffering du modèle PLCopen Motion Control, partie 4 (cf. section 3.1.1.1).
— un mode de déplacement de type *suivi de consigne de position périodique* (`AUTOMD_CYCLICPOS`) qui va permettre de mettre en place tous les modes de contrôles évolués, avec des algorithmes d'interpolation présents dans les blocs fonctionnels de pilotage de robot (comme la fonction d'interpolation dans l'espace opérationnel), soit avec des générateurs de consigne externes, exécutés dans l'OS temps-réel ou depuis un intergiciel tiers.

3.3.2 Applications

Cette section présente des exemples d'implémentation de notre architecture générique sur des mécanismes spécifiques. Pour ces deux exemples, l'intégration du logiciel de contrôle-commande

au plus près des contrôleurs de mouvement est nécessaire pour atteindre les performances attendues.

3.3.2.1 Pilotage d'une roue à inertie variable

A titre d'exemple, la fonctionnalité *pilotage de mécanisme* est utilisée pour le contrôle-commande d'un système mécatronique conçu et réalisé dans l'équipe RoBioSS par Antoine Eon et Arnaud Decatoire dans le cadre d'une prestation pour le CRITT Sport et Loisirs de Châtellerault.

Il s'agit d'une roue à inertie variable contrôlée destinée à faire comprendre à des enfants des concepts physiques à partir d'informations proprioceptives. Un sujet assis sur un tabouret en liaison pivot d'axe vertical avec le sol, tient devant lui, bras tendus, une roue d'inertie par deux poignées sur l'axe de rotation de la roue. La vitesse de rotation et la valeur du moment d'inertie de la roue sont variables et contrôlées par le sujet qui va pouvoir expérimenter les principes de conservation du moment cinétique et d'effet gyroscopique.

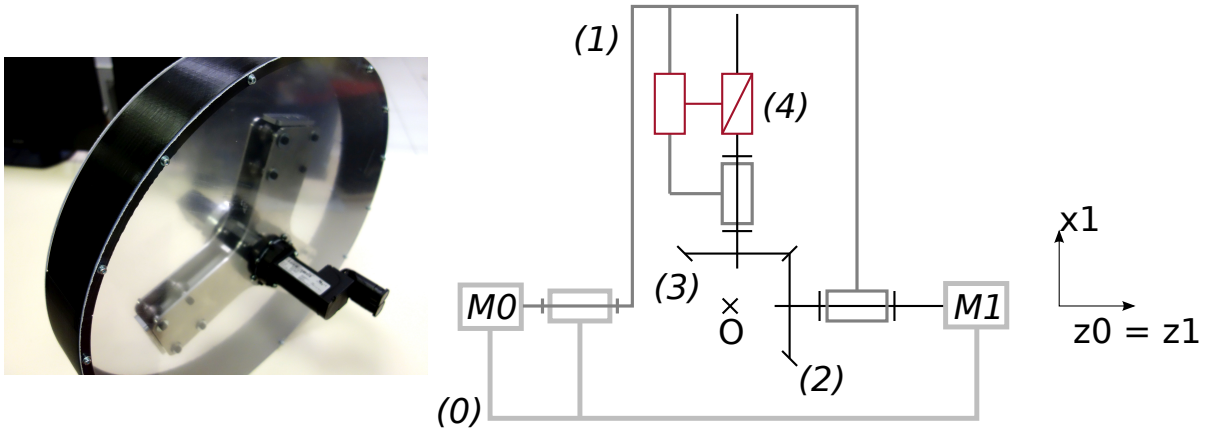


FIGURE 3.15 – Roue à inertie variable et schéma cinématique du mécanisme.

La roue présentée sur la figure 3.15 est motorisée par deux petits moteurs Brushless B&R 8LVA13 commandés par un contrôleur de mouvement ACOPOSmicro qui sont placés dans les poignées que tient le sujet. L'un des moteurs réalise l'entraînement de la roue, le second permet de déplacer radialement trois masses pour faire varier le moment d'inertie de l'ensemble. Le schéma cinématique décrit la transmission de puissance dans le mécanisme. Il permet de paramétrer les relations de couplage entre les rotations moteurs et les sorties du mécanisme, c'est à dire la rotation de la roue et la translation radiale des masses.

La tâche de gestion machine va sélectionner et coordonner les fonctions à utiliser dans ce cas précis. Ici, l'axe d'entraînement de la roue est piloté par paliers de vitesse réglables et la position des masses est définie par des paliers de position. La prise d'origine, dans le cas de ce mécanisme couplé avec des contraintes d'encombrement fortes, est réalisée par une fonction spécifique. La prise d'origine s'effectue sur une butée physique détectée par le dépassement d'un seuil de couple des moteurs.

Pour réaliser le logiciel de contrôle commande, il n'est pas possible d'utiliser directement l'unité d'organisation de programme de contrôle de mécanismes présentée sur la figure 3.12. En effet ce système demande des modes de pilotage différents pour les deux axes articulaires :

l'axe d'entraînement est piloté par des consignes de vitesses et l'axe de réglage de l'inertie, par des consignes de position. En outre, en raison de l'irréversibilité des liaisons vis/écrous et du frottement important dans le réducteur du moteur 1, la coordination des deux axes moteurs doit être très précisément réalisée pour ne pas bloquer le mécanisme.

Pour ces raisons, une classe spéciale de commande a été réalisée pour ce mécanisme. Elle exécute des blocs fonctionnels complémentaires de PLCopen Motion Control partie 1 pour confier au gestionnaire d'axe la coordination des deux axes moteurs. Il s'agit des blocs fonctionnels `MC_Gear` et `MC_Phasing`. Le premier crée une relation de couplage par un rapport de vitesses et le second permet d'ajouter un déphasage autour de cette relation.

Cet exemple permet d'illustrer l'intérêt d'adopter le standard PLCopen Motion Control pour pouvoir exploiter, lorsque le besoin émerge, des fonctions de mouvement complexes disponibles avec tous les équipements conformes au standard. Dans ce cas, un bénéfice complémentaire est que le transfert des fonctions de coordination entre moteurs à la carte de l'électronique de commande permet de choisir un calculateur central de plus faible puissance de calcul.

3.3.2.2 Extension au pilotage en couple articulaire

La figure 3.16 présente le banc d'apprentissage du geste *SHIVA* développé dans l'équipe RoBioSS par Arnaud Decatoire et Antoine Eon. Ce dispositif permet d'évaluer les stratégies d'apprentissage d'un geste défini par une séquence temporelle d'actions mécaniques à réaliser. Les premières applications de ce système portent sur l'amélioration des performances de sportifs et sur des méthodes de rééducation progressive, personnalisée à la morphologie et à l'état d'une personne blessée ou malade.

Le banc SHIVA est motorisé par un actionneur instrumenté également développé par Antoine Eon qui a conçu une gamme d'actionneurs électriques compacts et configurables pour ajuster la zone de fonctionnement couple / vitesse en fonction des exigences d'une application. Ces dispositifs appelés *ARM* (Actionneurs Robotiques Modulaires) sont constitués d'un moteur synchrone sans balais, d'un réducteur Harmonic Drive à rapport de réduction élevé et d'un capteur de couple en sortie de réducteur. Le choix de l'association moteur et réducteur permet de couvrir une large plage de spécifications. Le capteur de couple est intégré sur la bride de sortie du réducteur. Il est dimensionné en fonction du réducteur pour offrir la meilleure résolution possible avec la technologie retenue. Ce capteur est constitué d'un groupe de jauges de déformation dont le signal est conditionné par une carte B&R X20AI1744 présentée au chapitre 2.3.1. L'usage de ce capteur permet d'envisager la réalisation d'une commande en couple à partir de la mesure du couple transmis en sortie de réducteur. Ce travail est en cours de réalisation.

Sur le plan de l'architecture logicielle, cette nouvelle fonctionnalité impose une modification de la logique de couplage et de prise d'origine des groupes d'axes. La comparaison des stratégies de pilotage en position et en couple est présentée sur la figure 3.17. Dans le second cas, les axes moteurs sont contrôlés par des consignes de couple générées périodiquement à partir des couples articulaires désirés. Les axes moteurs sont les axes maîtres et les axes articulaires sont placés en suivi de consigne de position.

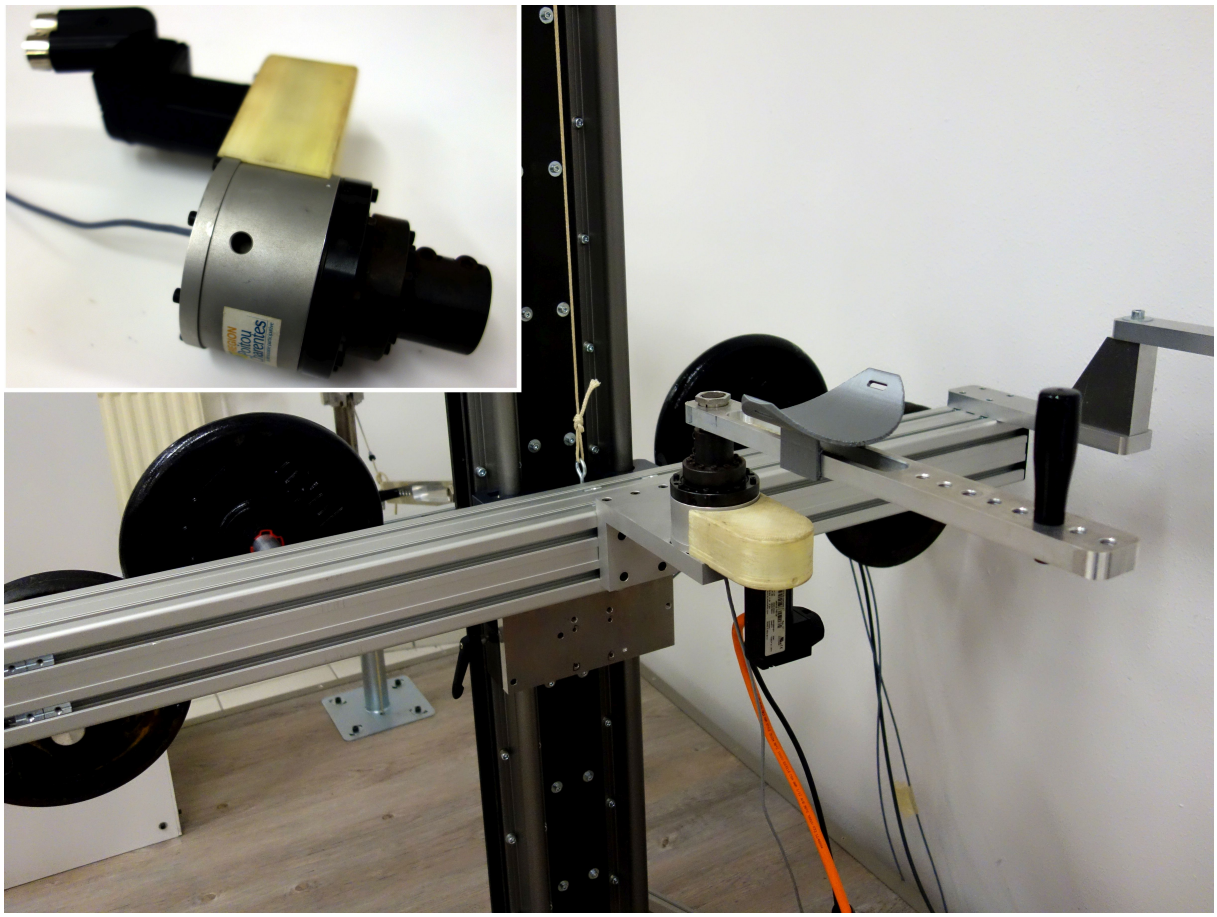


FIGURE 3.16 – Axe robotique modulaire (ARM) et intégration sur le banc SHIVA d'apprentissage du geste.

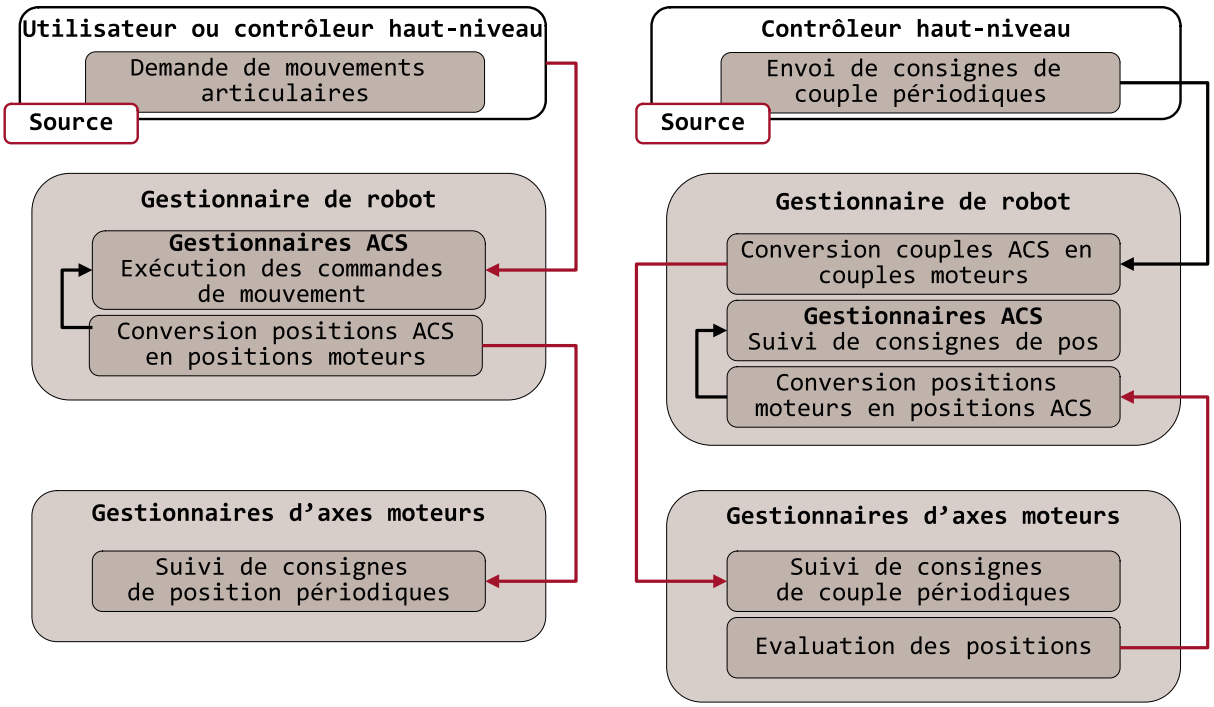


FIGURE 3.17 – Logique de couplage entre axes moteurs et axes articulaires suivant les modes de pilotage (gauche : pilotage en position, droite : pilotage en couple).

3.4 Pilotage des robots

Selon la décomposition que nous avons retenue, les systèmes de type robot se distinguent des mécanismes par l'existence d'un espace de travail complémentaire dans lequel les mouvements ou les tâches à réaliser sont définis. L'unité d'organisation de programme associée est décrite sur la figure 3.18, elle complète celle qui décrit le comportement des mécanismes en ajoutant des éléments de structure et des fonctions pour traiter le changement entre les espaces articulaire et opérationnel.

Pour pouvoir décrire la diversité des architectures que nous souhaitons pouvoir contrôler, nous distinguons les représentations suivantes :

— un groupe de $nMot$ moteurs à piloter,
— un groupe de $nAcs$ articulations pilotées qui sont liées aux moteurs par la transmission de puissance,
— un ensemble de $nJnt$ articulations entraînées qui peuvent être plus nombreuses que les articulations pilotées,
— un groupe de $nMcsFrames$ repères associés à un ou plusieurs effecteurs,
— un ensemble de $nMcsDof$ degrés de libertés contrôlables dans l'espace opérationnels.

Les fonctions d'information `fkm()` et `updateMcsInfo()` permettent d'obtenir les positions et orientations des repères des effecteurs, leurs torseurs cinématiques et les matrices Jacobiennes associées qui relient les vitesses articulaires et les vitesses opérationnelles.

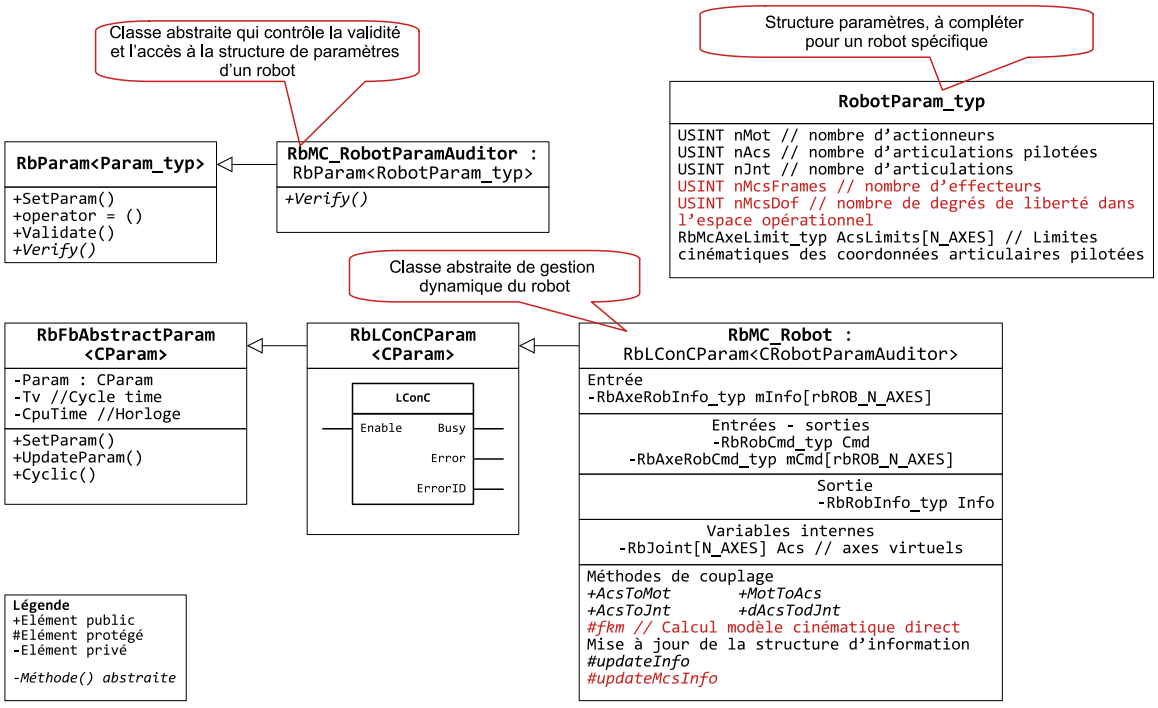


FIGURE 3.18 – Classes de base des modèles de robot dans `rtrmac`.

Les stratégies de pilotage dans l'espace opérationnel implémentées pour l'instant sont présentées sur la figure 3.19. La définition des tâches à accomplir par le robot doit être formalisée

en termes de vitesse opérationnelle. Il y a deux possibilités de définition des consignes :

— des consignes instantanées de vitesses des degrés de liberté pilotables peuvent être générées par un contrôleur haut-niveau, embarqué dans le système de contrôle-commande du robot ou exécuté sur un calculateur externe (*source 1*),

— un déplacement vers un point d'arrêt dans l'espace opérationnel avec interpolation à vitesses et accélérations saturées sur les degrés de liberté pilotables (*source 2*).

La seconde option utilise un interpolateur pour générer les consignes de vitesses instantanées.

A partir des vitesses opérationnelles et des informations d'états du robot, les vitesses articulaires sont calculées par la résolution d'un problème d'optimisation sous contraintes qui inverse au mieux le modèle cinématique en assurant le respect de contraintes sur les vitesses articulaires. Cette démarche est présentée à la section 3.5.4.

La section suivante présente la démarche de construction d'un modèle de robot à partir d'une architecture mécanique simple.

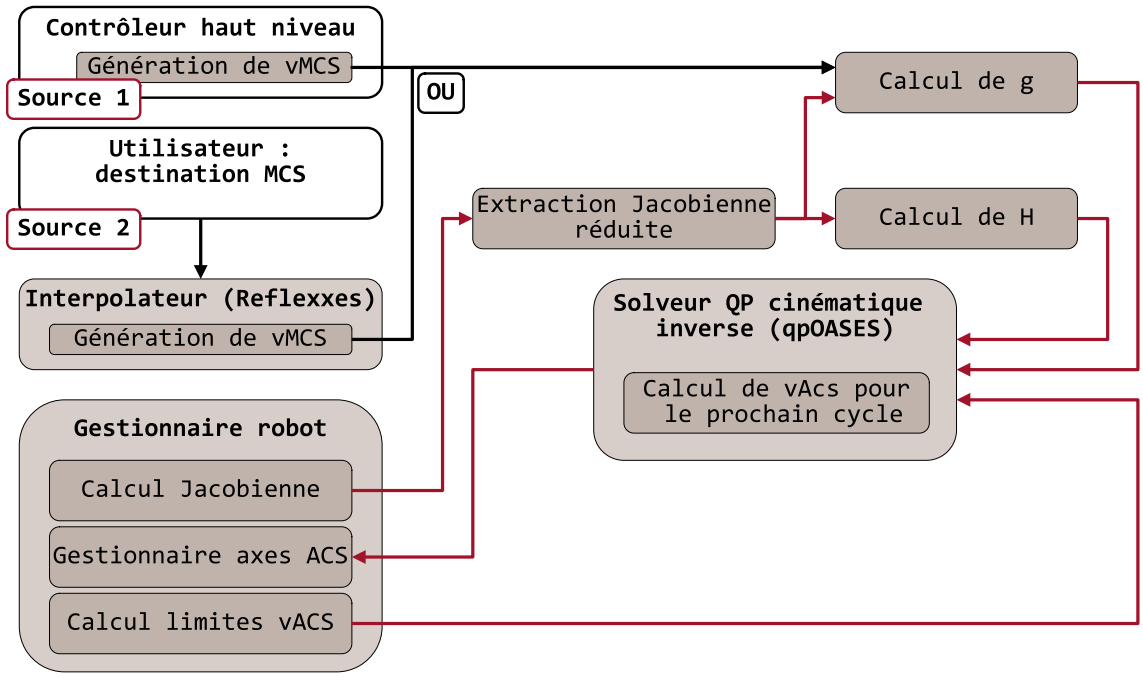


FIGURE 3.19 – Stratégies de pilotage dans l'espace opérationnel.

3.5 Illustration : contrôle-commande d'un robot sériel 2R

Pour valider l'approche proposée ici, j'ai implémenté la structure de contrôle sur un système robotique développé pour l'enseignement en licence professionnelle Automation et Robotique à l'université de Poitiers. Il s'agit d'un robot sériel à deux degrés de liberté en rotation dont la structure est légère, les deux moteurs étant fixés sur le bâti et qui ne présente pas de limite articulaire. Ce mécanisme présente un couplage dans la transmission ce qui nous permettra de

valider les fonctions de passage entre l'espace des moteurs et l'espace articulaire.

La figure 3.20 présente les composants de l'armoire de commande. Il s'agit de composants standards qui vont permettre d'évaluer la possibilité d'exécuter les logiciels que nous développons sur des architectures industrielles. Tous les contrôleurs de mouvement B&R ont la même interface logicielle avec la bibliothèque PLCopen Motion Control qui donne accès aux fonctions présentées à la section 2.1.4.1. La mise à l'échelle vers des machines de plus grandes puissances et présentant plus d'axes de mouvement est donc tout à fait directe.

La description du robot s'appuie sur les repères suivants :

— base : $R_0 = (O_1, (x_0, y_0, z))$
— bras 1 : $R_1 = (O_1, (x_1, y_1, z))$
— bras 2 : $R_2 = (O_2, (x_2, y_2, z))$
— l'effecteur est représenté par la position du Tool Center Point T et le repère associé $R_3 = (T, (x_3, y_3, z))$

Le paramétrage du système est défini dans trois espaces de dimension deux par :

— dans l'espace des moteurs, les positions angulaires m_1, m_2,
— dans l'espace articulaire : les positions angulaires $q_1 = (\vec{x}_0, \vec{x}_1)$, $q_2 = (\vec{x}_1, \vec{x}_2)$
— dans l'espace opérationnel : la position du `TCP` dans le plan (O_1, x_0, y_0).

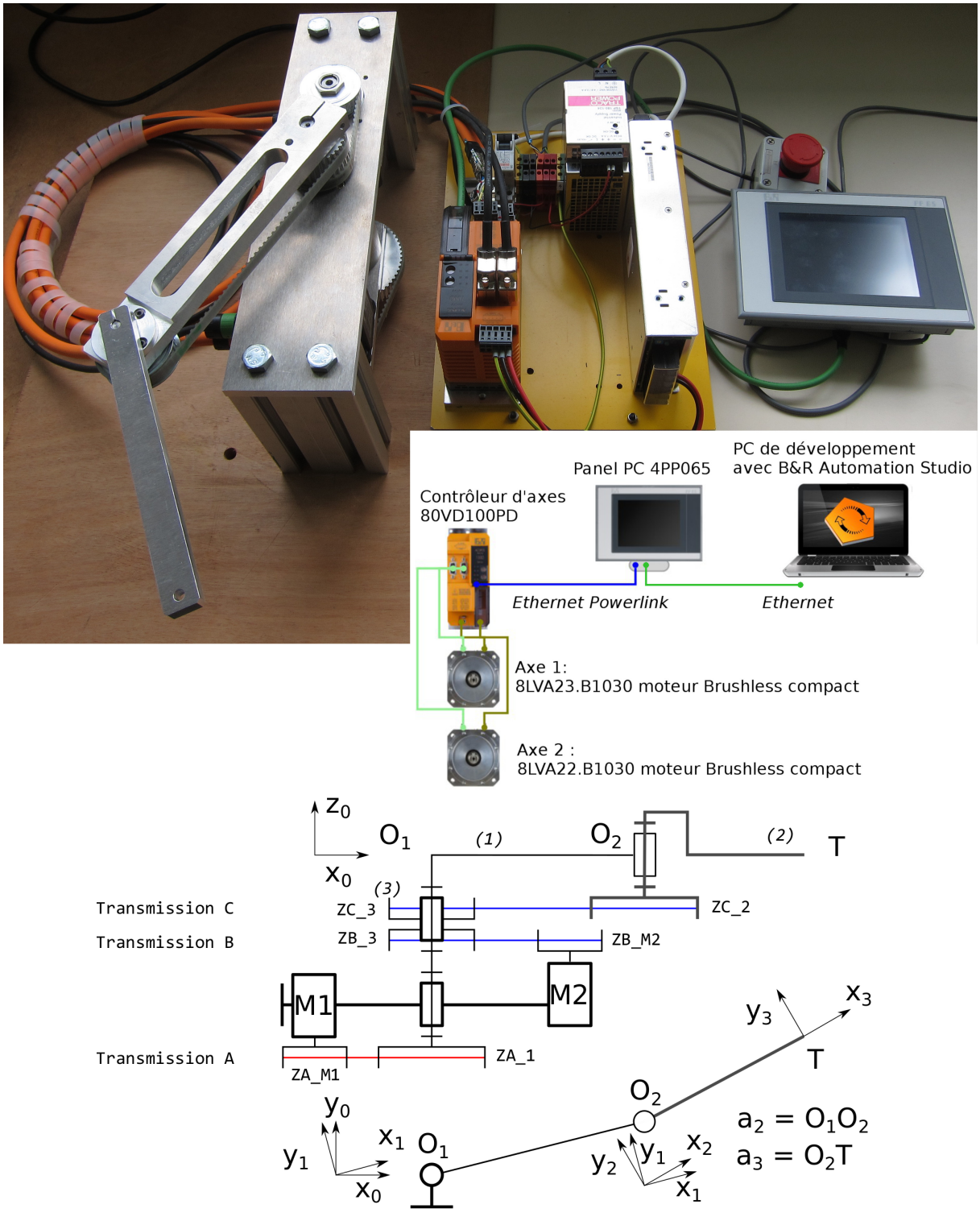


FIGURE 3.20 – Robot 2R sériel, vue du système et paramétrage.

3.5.1 Description de la partie mécanisme

Ce mécanisme présente deux moteurs (dont les positions angulaires sont notées m_1 et m_2), deux articulations pilotées (q_1 et q_2) et trois étages de transmission, dont les rapports de transmission sont définis par les nombres de dents des poulies associées :

— `A` : entre le moteur 1 et l'axe 1, poulies de nombre de dents Z_{M1}, Z_{1A},
— `B` : entre le moteur 2 et la poulie intermédiaire (3), paramètres Z_{M2}, Z_{3B}
— `C` : entre la poulie intermédiaire (3) et le bras (2), paramètres Z_{3C}, Z_2

Les relations de couplage directes en vitesse sont les suivantes :

$$\dot{q}_1 = \frac{Z_{M1}}{Z_1}\dot{m}_1$$

$$\dot{q}_2 = \frac{Z_{3C}}{Z_2}\left(\frac{Z_{M2}}{Z_{3B}}\dot{m}_2 - \frac{Z_{M1}}{Z_1}\dot{m}_1\right)$$

Les relations de couplage inverses sont :

$$\dot{m}_1 = \frac{Z_1}{Z_{M1}}\dot{q}_1 \tag{3.1}$$

$$\dot{m}_2 = \frac{Z_{3B}}{Z_{M2}}\left(\frac{Z_2}{Z_{3C}}\dot{q}_2 + \dot{q}_1\right) \tag{3.2}$$

$$\tag{3.3}$$

Ces relations sont implémentées dans les méthodes `MotToAcs` et `AcsToMot` de la classe de contrôle de ce robot. Une fois le démarrage et la prise d'origine réalisés, les axes moteurs deviennent esclaves des axes virtuels liés aux articulations, les consignes positions moteurs sont calculées à chaque instant par l'exécution de la méthode `AcsToMot`.

3.5.2 Description de la partie robot

Le paramétrage associé au robot est illustré sur la figure 3.20. Les matrices de passage homogènes entre les repères associés aux différents solides sont définies ci-dessous. Les notations simplifiées $C1$, $S1$ représentent $cos(q_1)$ et $sin(q_1)$, les constantes a_1, a_2 sont les dimensions caractéristiques du système.

$$T^{01} = \begin{pmatrix} C1 & -S1 & 0 & 0 \\ S1 & C1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T^{12} = \begin{pmatrix} C2 & -S2 & 0 & a_2 \\ S2 & C2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T^{23} = \begin{pmatrix} 1 & 0 & 0 & a_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Les valeurs des paramètres géométriques du robot sont présentées dans la table 3.2.

TABLE 3.2 – Éléments caractéristiques du mécanisme et du robot.

ZA_{M1}	ZA_1	ZB_{M2}	ZB_3	ZC_2	ZC_3	a_2	a_3
30	60	30	32	31	32	202.24 mm	180 mm

3.5.3 Modèle cinématique

La matrice jacobienne naturelle J relie la vitesse absolue du Tool Center Point (le point de référence du repère outil) du solide (2) et la vitesse de rotation absolue de (2) aux vitesses articulaires. Elle peut être obtenue à partir du calcul des torseurs cinématiques des liaisons et par la relation de composition des vitesses.

Liaison 1 : pivot d'axe $(O_1, \vec{z}_0)$

$$\{C_{1/0}\} = \left\{ \begin{matrix} \vec{\Omega}_{1/0} = \dot{q}_1 \vec{z_0} \\ \vec{0} \end{matrix} \right\}_O$$

avec $\vec{O_1T} = (a_2.C1 + a_3.C12)\vec{x}_0 + (a_2.S1 + a_3.S12)\vec{y}_0$

$$\vec{V}_{T,1/0} = \vec{V}_{O_1,1/0} + \vec{\Omega}_{1/0} \wedge \vec{O_1T} = \dot{q_1}\left(-(a_2.S1 + a_3.S12)\vec{x}_0 + (a_2.C1 + a_3.C12)\vec{y_0}\right)$$

Liaison 2 : pivot d'axe $(O_2, \vec{z}_0)$

$$\{C_{2/1}\} = \left\{ \begin{matrix} \vec{\Omega}_{2/1} = \dot{q}_2 \vec{z_0} \\ \vec{0} \end{matrix} \right\}_{O_2}$$

avec $\vec{O_2T} = a_3.C12.\vec{x}_0 + a_3.S12.\vec{y}_0$

$$\vec{V}_{T,2/1} = \vec{V}_{O_2,2/1} + \vec{\Omega}_{2/1} \wedge \vec{O_2T} = \dot{q}_2\left(-a_3.S12.\vec{x}_0 + a_3.C12.\vec{y_0}\right)$$

Finalement, la matrice J telle que $(\vec{\omega}_{2/0}, \vec{v}_{T,2/0})^T_{R_0} = J \cdot (\dot{q}_1, \dot{q}_2)^T$ est :

$$J = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 1 \\ -(a_2.S1 + a_3.S12) & -a_3.S12 \\ a_2.C1 + a_3.C12 & a_3.C12 \\ 0 & 0 \end{pmatrix}$$

3.5.4 Pilotage dans l'espace opérationnel

Comme introduit dans la section 3.4, nous avons mis en place deux stratégies de pilotage en position dans l'espace opérationnel. Les deux approches partagent le même algorithme d'inversion du modèle cinématique pour calculer des consignes de vitesses articulaires instantanées. La figure 3.21 décrit la formalisation de ce problème. Les axes articulaires sont contrôlés par une consigne de vitesse qui doit être obtenue à partir des mouvements désirés dans l'espace

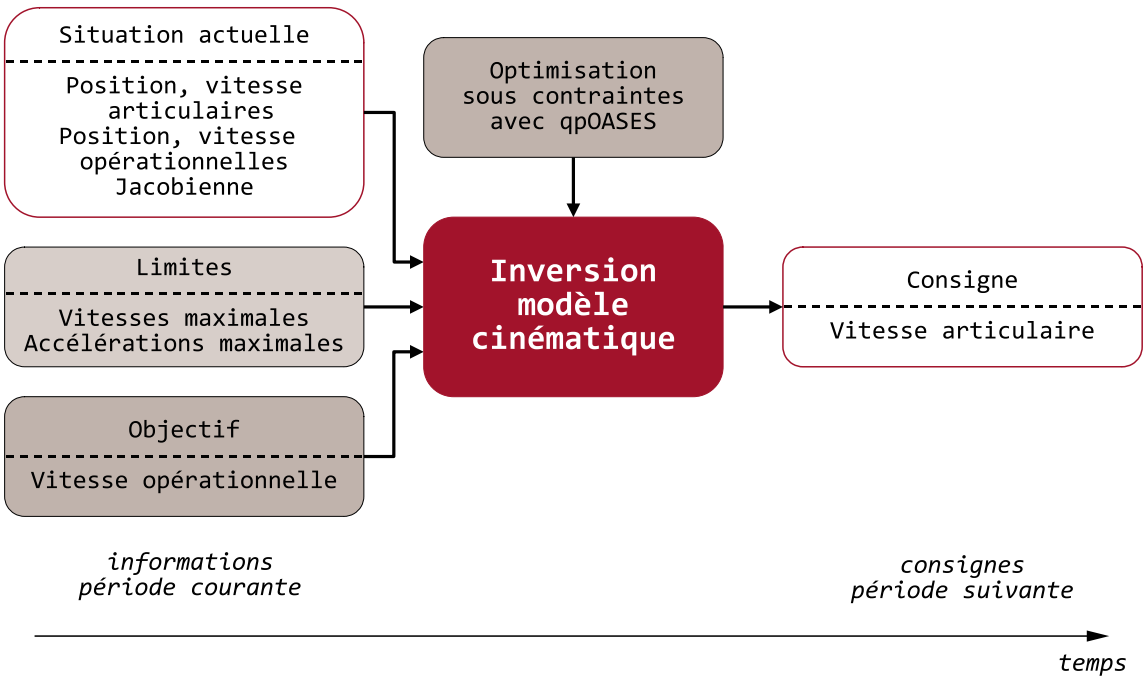


FIGURE 3.21 – Inversion modèle cinématique.

opérationnel. Celle-ci est décrite sous la forme du torseur cinématique absolu de l'effecteur ou des vitesses instantanées de certains degrés de libertés dans l'espace opérationnel, dans le cas de robots pour lesquels l'effecteur ne dispose pas de six degrés de liberté.

Dans l'objectif de disposer d'un système de contrôle-commande robuste et pouvant réagir rapidement à des changements de consigne, nous avons choisi de formaliser la stratégie de calcul des vitesses articulaires sous la forme d'un problème d'optimisation sous contraintes qui rentre dans le cadre de la programmation quadratique.

Les données d'entrée du problème sont :

— la situation du robot à l'instant t_k définie par ses positions et vitesses articulaires (q_k et $\dot{q}_k$), la matrice jacobienne calculée à cet instant, J,
— les vitesses opérationnelles désirées à l'instant suivant : $\dot{x}_k^c$, fournies par l'algorithme de génération de trajectoire ou par la traduction d'une commande utilisateur dans le cas d'un mode téléopéré,
— un ensemble de limites actualisables à chaque instant portant sur les positions, vitesses ou accélérations articulaires et traduites en limites sur les vitesses articulaires.

La résolution du problème d'optimisation sous contraintes fournit les vitesses articulaires qui seront traitées comme consignes pour l'actionnement des articulations.

Ces vitesses $\dot{q}$ sont la solution de :

$$\dot{q} = argmin_{\dot{q}} \left(\frac{1}{2} \| J\dot{q} - \dot{x}^c \|^2 \right)$$

$$\dot{q} = argmin_{\dot{q}} \left(\frac{1}{2} \dot{q}^T J^T J \dot{q} - \dot{q}^T J^T \dot{x}^c \right)$$

Si les contraintes portent sur le respect des limites d'accélération et de vitesse des articulations, elles peuvent s'exprimer de la façon suivante :

$$-\dot{q}_{max} \leq \dot{q}_{k+1} \leq \dot{q}_{max}$$

$$q_{min} \leq q_{k+1} \leq q_{max} \Rightarrow \frac{q_{min} - q_k}{t_C} \leq \dot{q}_{k+1} \leq \frac{q_{max} - q_k}{t_C}$$

soit :

$$max(-\dot{q}_{max}, \frac{q_{min} - q_k}{t_C}) \leq \dot{q}_{k+1} \leq min(\dot{q}_{max}, \frac{q_{max} - q_k}{t_C})$$

La bibliothèque qpOASES résout des problèmes quadratiques mis sous la forme suivante :

$$min_x \left(\frac{1}{2} x^T H x + x^T g \right)$$

avec :

$$lbA \leq Ax \leq ubA$$

$$lb \leq x \leq ub$$

— H, matrice Hessienne : $H = J^T J + \omega_0 \mathbb{1}$
— g, vecteur gradient : $g = -J^T \dot{x}_c$
— lb, ub, vecteurs limites : $lb = max(-\dot{q}_{max}, \frac{q_{min} - q_k}{t_C})$, $ub = min(\dot{q}_{max}, \frac{q_{max} - q_k}{t_C})$

Pour les architectures robotiques considérées pour l'instant, les problèmes quadratiques rencontrés ont les propriétés suivantes :

— la dimension du problème d'optimisation à résoudre est faible : le nombre de variables est de l'ordre d'une dizaine
— la matrice Hessienne et le vecteur gradient changent à tous les cycles automates,
— les bornes sur les inconnues peuvent varier suivant les tâches à remplir,
— le problème ne présente pas de contraintes couplées. En cas de mécanisme présentant un sous-actionnement, le respect des bornes sur les articulations contrôlées garantit, par construction, le respect des limites en déplacement sur les articulations couplées.

Sous ces hypothèses, la résolution du problème d'inversion du modèle cinématique se formule de la façon suivante :

1. calculer la matrice Hessienne $H = J^T.J$
2. calculer le vecteur gradient $g = -J^T.\dot{x}$
3. calculer les bornes inférieure lb et supérieure ub sur $\dot{q}$
4. obtenir la nouvelle consigne de vitesse articulaire $\dot{q}_{k+1}$ telle que :

$$min_{\dot{q}} \left(\frac{1}{2} \dot{q}^T H \dot{q} + \dot{q}^T.g \right)$$

avec :

$$lb \leq \dot{q} \leq ub$$

3.5.5 Suivi d'une consigne interpolée dans l'espace opérationnel

Pour illustrer le fonctionnement de la méthode de pilotage présentée précédemment, nous allons présenter un exemple de suivi de trajectoire. Il s'agit d'une succession de mouvements en ligne droite dans l'espace opérationnel. Les points de passage ont été choisis de telle sorte que la trajectoire de consigne du centre outil passe par des zones en dehors de l'espace de travail. Les deux bras du robot étant de longueurs L_1 et L_2 différentes, la surface contenant l'ensemble des positions accessibles est une couronne de rayon externe $L_1 + L_2$ et de rayon interne $L_1 - L_2$.

Deux exemples de stratégie de génération de la trajectoire de référence sont présentés [1] :

— la trajectoire est interpolée entre deux points d'arrêt avec des limites en vitesse et accélération sur l'abscisse curviligne,

— la trajectoire est construite à chaque instant à partir d'un objectif à atteindre avec des limites en vitesses et en accélérations selon les deux axes du plan à partir de la situation instantanée $(x, \dot{x})$, avec la bibliothèque *Reflexxes type II*.

La figure 3.22 présente la trajectoire de référence avec le premier mode. Le centre outil suit le trajet $P_1 \to P_2 \to P_3 \to P_1$. Les segments $P_1 \to P_2$ et $P_3 \to P_1$ passent par des zones non accessibles. L'inversion directe du modèle cinématique n'a pas de solution à ces endroits et génère des erreurs d'exécution du code. Le problème d'optimisation sous contraintes fournit à chaque instant une solution approchée en vitesses articulaires compatible avec les limites spécifiées ou impose des vitesses nulles.

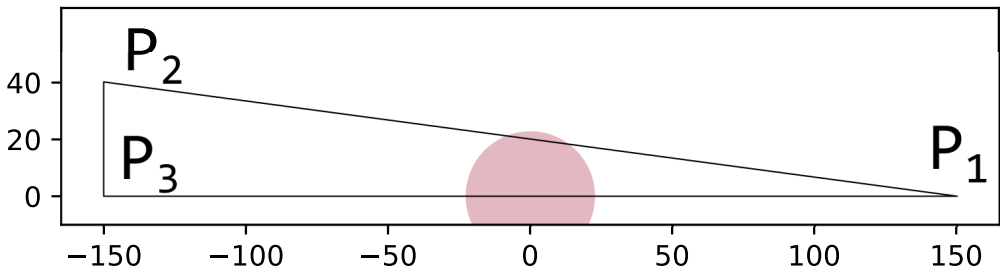


FIGURE 3.22 – Trajectoire de référence dans le plan (x, y). Les dimensions sont en mm. Le disque correspond à une zone hors espace de travail.

Les figures 3.23 et 3.24 présentent les mouvements réalisés avec les deux modes d'interpolation. Le passage des obstacles s'effectue sans arrêt du contrôleur avec des oscillations importantes des vitesses articulaires entre les valeurs de contraintes imposées. Les figures présentent :

— la trajectoire de consigne et la trajectoire réalisée,

— la position et la vitesse du centre outil dans le plan de déplacement,

— l'évolution des consignes et l'écart entre les positions du robot et les consignes.

Pour améliorer le comportement du système, il est possible de modifier l'expression de la fonction objectif qui va limiter ces oscillations. Cette approche est appelée régularisation de Tikhonov, elle consiste à ajouter un terme qui tend à minimiser les valeurs maximales des vitesses angulaires. Cette approche est régulièrement utilisée, par exemple dans le cas de la robotique

1. Ces trajectoires sont illustrées en vidéo sur src/koda.cnrs.fr.

humanoïde où des systèmes avec des nombre de degrés de liberté élevés doivent répondre à des plusieurs objectifs parfois incompatibles [Escande 2014], [Calinon 2017].

Les vitesses articulaires $\dot{q}$ sont alors solutions de :

$$\dot{q} = argmin_{\dot{q}} \left(\frac{1}{2} \| J\dot{q} - \dot{x}^c \|^2 + \frac{1}{2} \omega_0 \| \dot{q} \| \right)$$

$$\dot{q} = argmin_{\dot{q}} \left(\frac{1}{2} \dot{q}^T (J^T J + \omega_0 \mathbb{1}) \dot{q} - \dot{q}^T J^T \dot{x}^c \right)$$

La matrice Hessienne de notre nouvelle fonction d'optimisation s'écrit :

$$H = J^T J + \omega_0 \mathbb{1}$$

avec ω_0, le coefficient de régularisation associé aux vitesses articulaires.

Les figures 3.25 et 3.26 présentent les nouveaux mouvements obtenus avec l'ajout du terme de régularisation de Thikonov. Les trajectoires sont lissées et les changements de signe des vitesses articulaires n'apparaissent plus. Dans le cas de l'interpolation directe, les consignes opérationnelles ne tiennent pas compte du retard pris par le robot lors du contournement de la zone inatteignable. Les consignes en sortie de l'interpolateur de la bibliothèque *Reflexxes* s'adaptent à la perturbation rencontrée. Dans les cas 3 et 4, où la régulation est activée, l'écart entre la consigne et la position réelle est presque nul pour l'interpolation par *Reflexxes*. Avec l'interpolation directe, cet écart augmente lorsque le robot prend du retard puis s'annule par le terme de correction intégré dans le calcul de la vitesse de consigne.

Une dernière amélioration peut être ajoutée. L'intérêt de minimiser les vitesses articulaires ne se présente que lorsque le robot s'approche de ses positions singulières. Comme le problème d'inversion du modèle cinématique est formalisé et résolu à chaque pas de temps, il est possible de pondérer le terme de régularisation en évaluant la proximité de la configuration du robot à ses configurations singulières. Deux critères qui évaluent cela ont été testés, ils portent sur l'état de la matrice jacobienne du robot. Les positions singulières sont atteintes lorsque l'une des valeurs propres de cette matrice s'annule. La figure 3.27 présente l'évaluation dans l'espace de travail du déterminant normalisé de la matrice Jacobienne et par le rapport de la seconde valeur singulière sur la première. Cette approche est illustrée dans le cas d'une architecture robotique où la solution est triviale, mais elle peut être généralisée à des systèmes plus complexes.

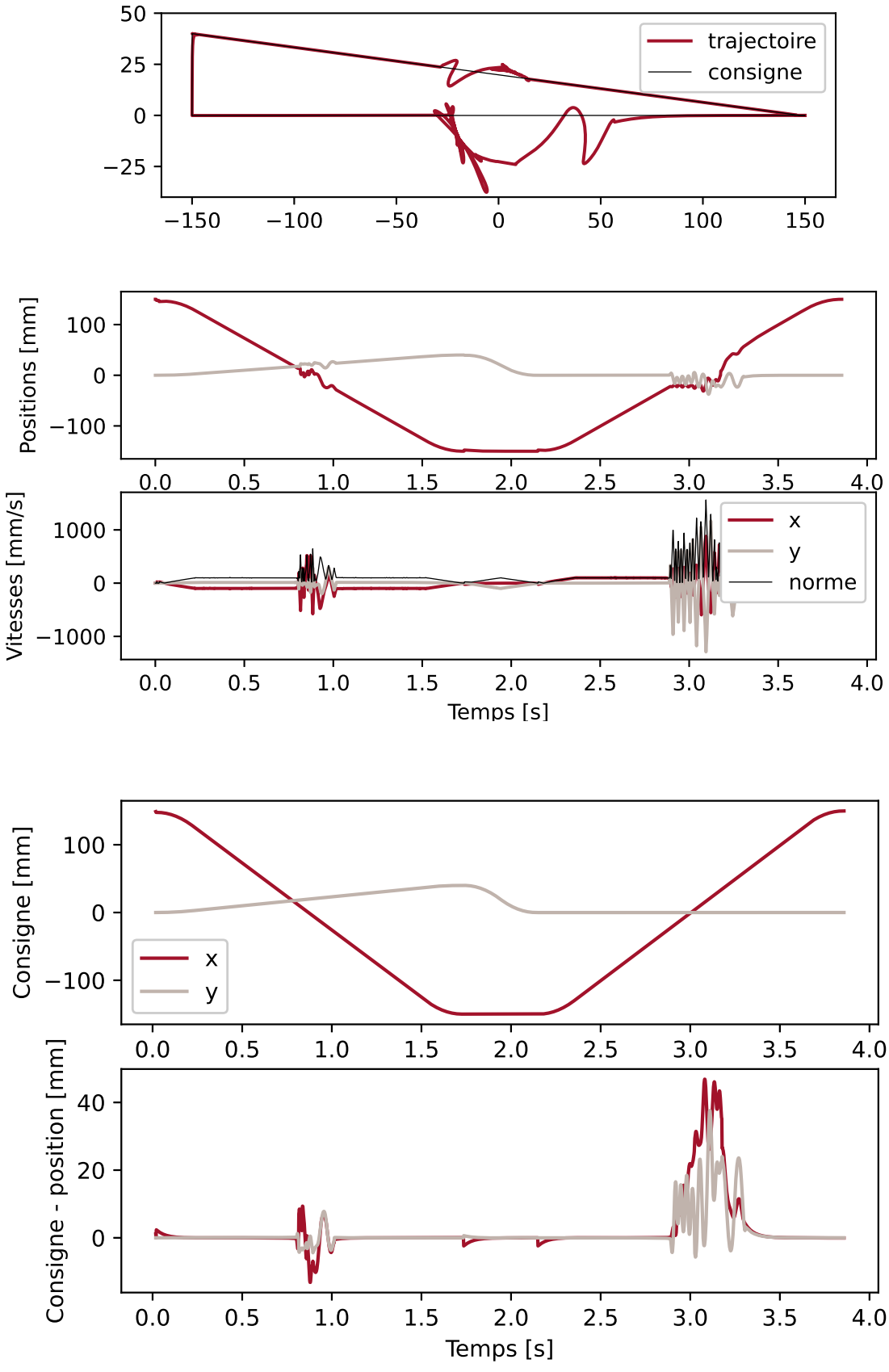


FIGURE 3.23 – Interpolation directe, inversion du modèle cinématique sans régularisation. Trajectoires de consigne et réalisées (haut), déplacement et vitesse dans l'espace opérationnel (milieu), écart entre les consignes et les positions réelles (bas).

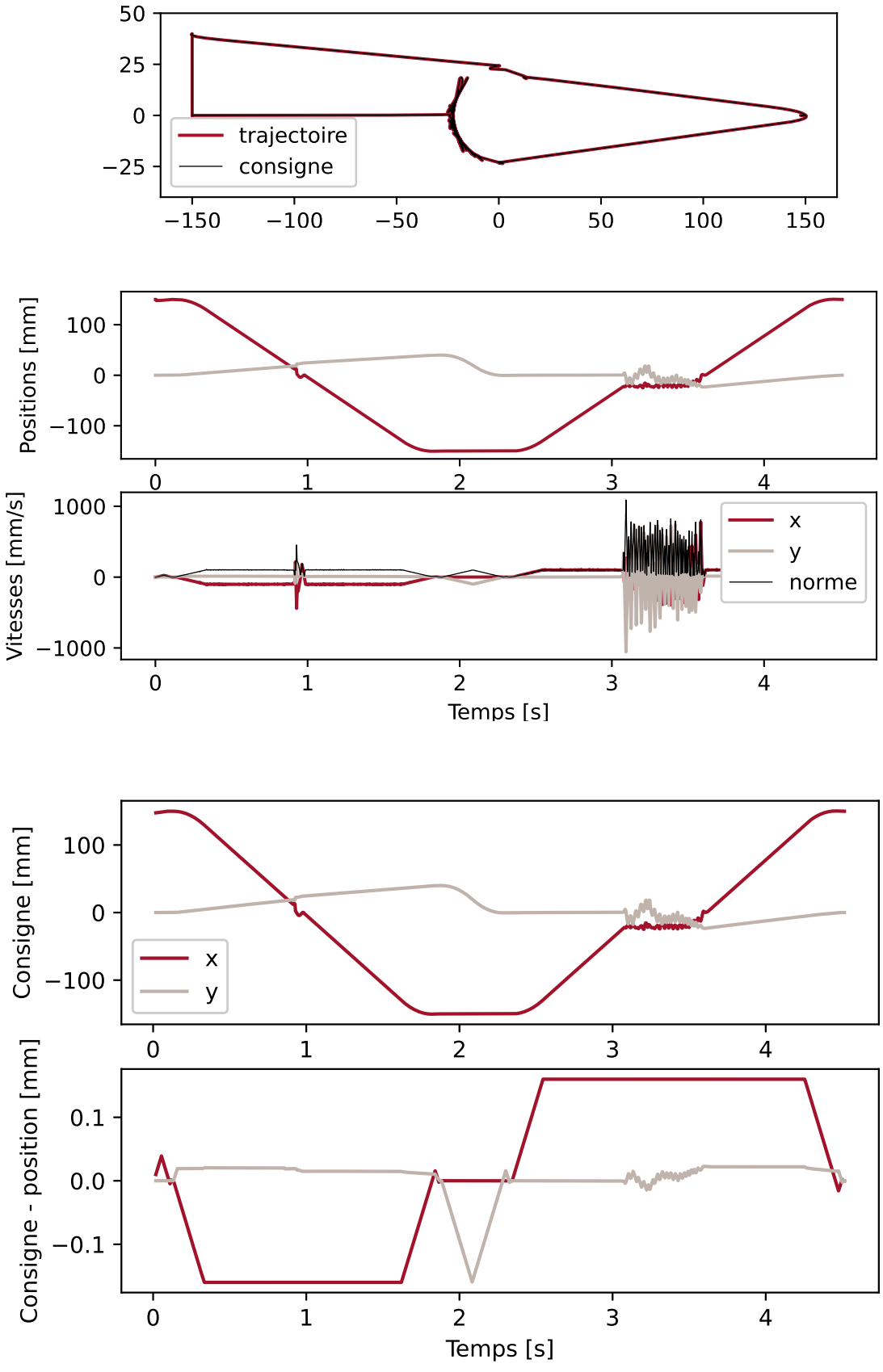


FIGURE 3.24 – Interpolation par *Reflexxes*, inversion du modèle cinématique sans régularisation.

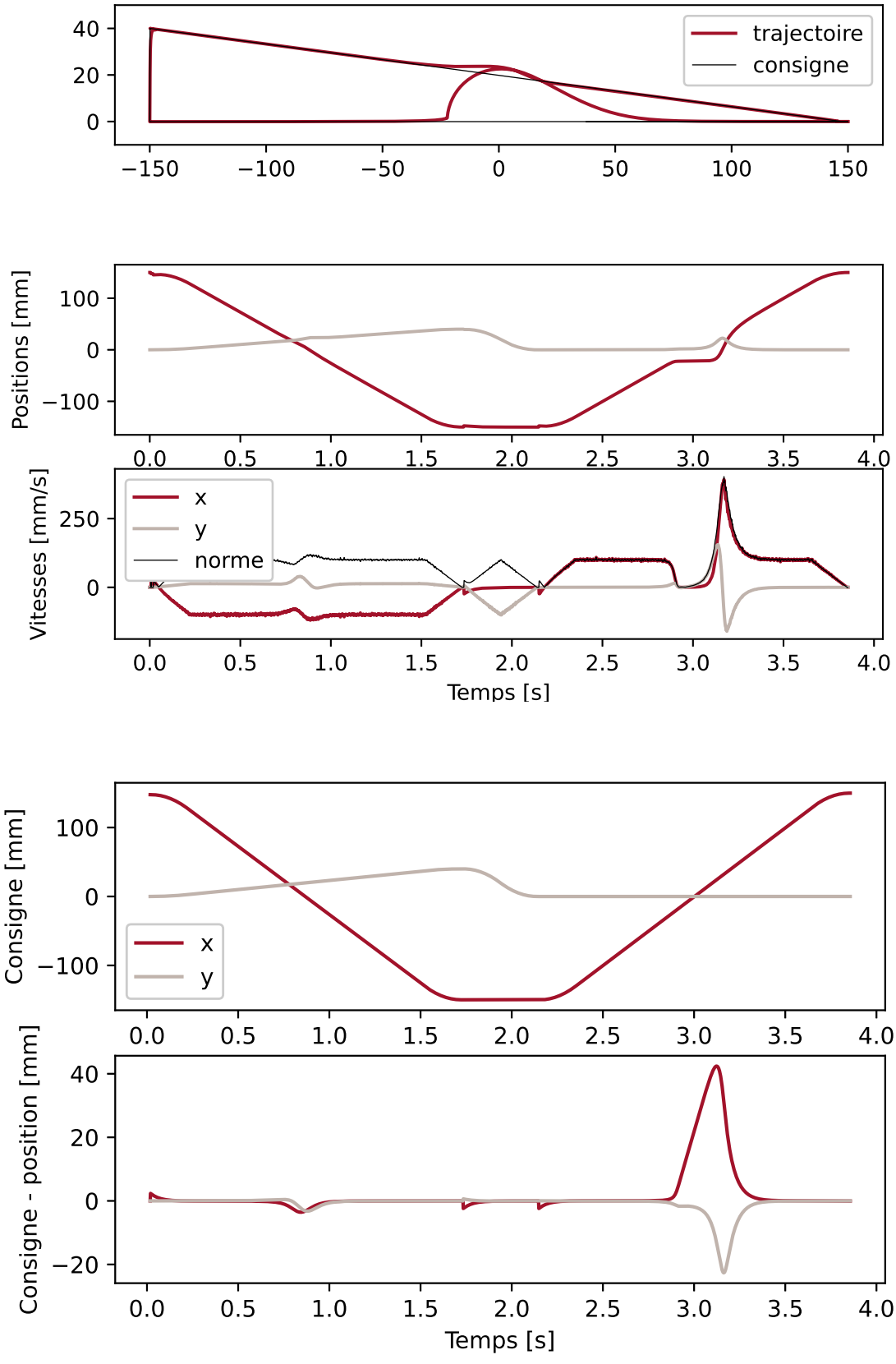


FIGURE 3.25 – Interpolation directe, inversion du modèle cinématique avec régularisation.

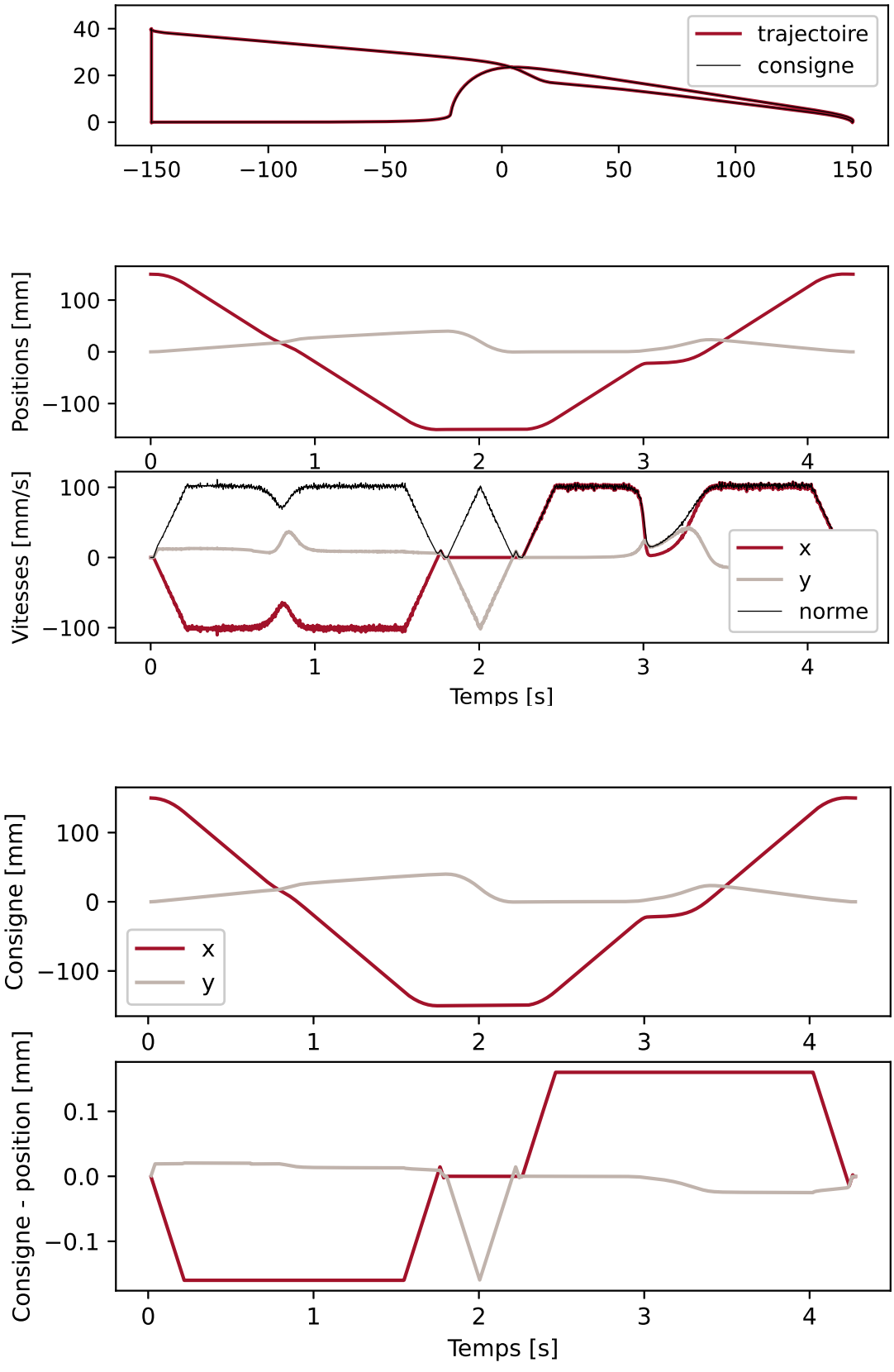


FIGURE 3.26 – Interpolation par *Reflexxes*, inversion du modèle cinématique avec régularisation.

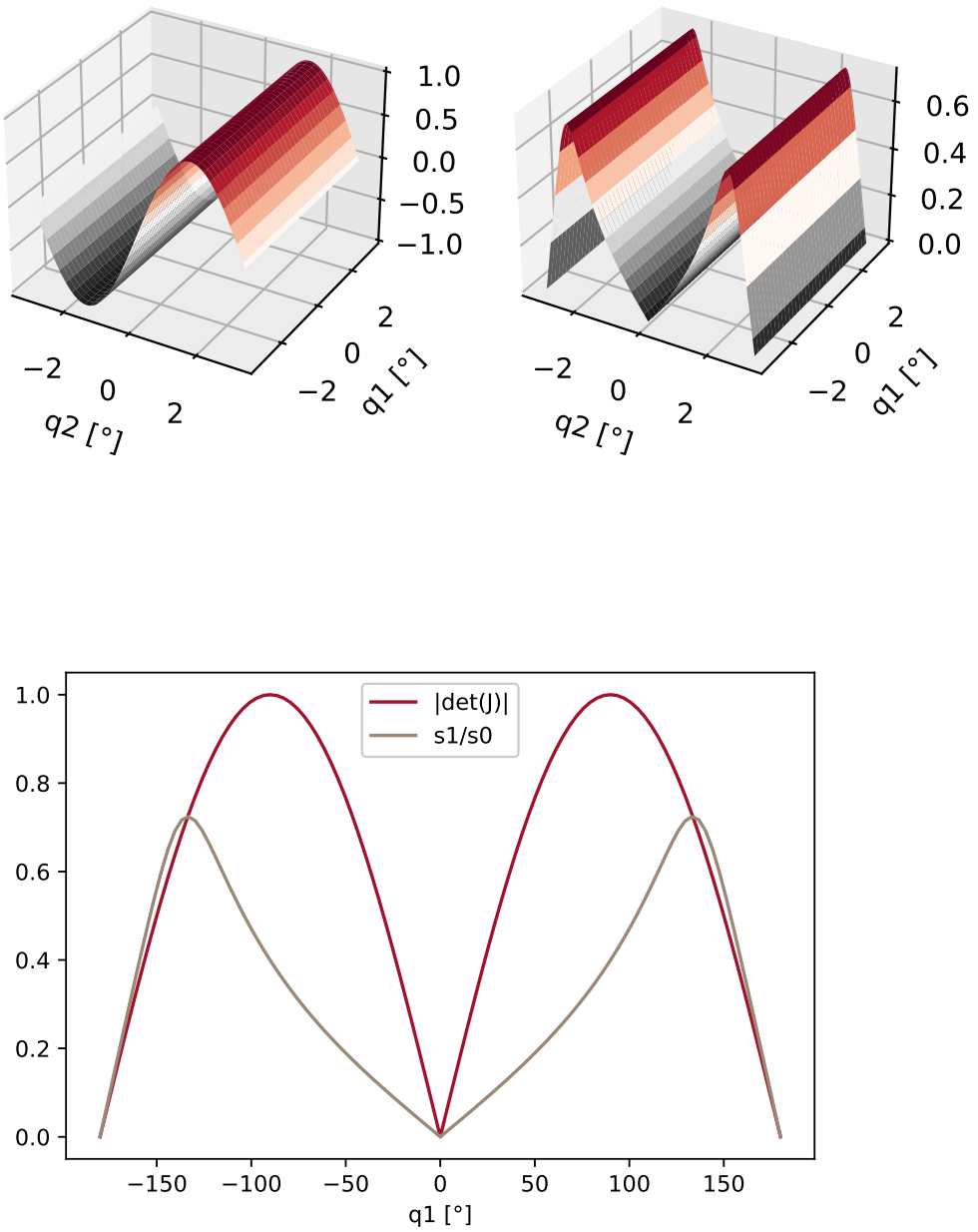


FIGURE 3.27 – Indicateurs de la proximité de positions singulières : déterminant normalisé de la matrice jacobienne et rapport des valeurs singulières de la matrice en fonction des positions articulaires. Haut : dépendance en (q_1, q_2), bas : évolution selon q_2.

3.5.6 Conclusion

Finalement, la stratégie de pilotage dans l'espace opérationnel implémentée permet de séparer fonctionnellement la génération de trajectoire et le calcul des consignes articulaires instantanées. La génération de trajectoire peut être :

- intégrée dans le logiciel de pilotage du robot dans le cas d'interpolation simple,
- calculée préalablement et chargée à partir d'un fichier,
- générée sur le PLC par une tâche spécifique au système à contrôler,
- générée sur un autre calculateur, à partir d'un jumeau numérique ou d'un système de téléopération.

Toutes ces stratégies peuvent fournir à chaque instant une consigne de position et de vitesse opérationnelle à atteindre à l'instant suivant. L'algorithme de calcul des consignes articulaires est alors défini ci-dessous.

Données d'entrées : $x^e_{i+1}, \dot{x}^e_{i+1}$ à atteindre au cycle suivant, situation courante du robot : position, vitesse articulaires : q_i, $\dot{q}_i$, position, vitesse opérationnelles : x_i, $\dot{x}_i$.

1. calculer la consigne de vitesse corrigée : $\dot{x}^c_{i+1} = \dot{x}^e + k_p(x - x^e)$
2. calculer la matrice Jacobienne J à partir de q_i, $\dot{q}_i$,
3. calculer le rapport entre les valeurs minimale et maximale des valeurs singulières s_k de J : $r_s = abs(min_k(s_k)/max_k(s_i))$
4. calculer la matrice Hessienne $H = J^T J - w_0 \mathbb{1}$ avec w_0 qui dépend de r_s,
5. calculer le vecteur gradient $g = -J^T \dot{x}^c_{i+1}$ à partir de la vitesse de consigne,
6. évaluer les limites sur $\dot{q}^e_{i+1}$ à partir des limites en accélération et vitesses des axes et éventuellement, en fonction de contraintes complémentaires liées à la tâche à réaliser,
7. déterminer la nouvelle vitesse de consigne $\dot{q}^e_{i+1}$ qui minimise $\frac{1}{2}\dot{q}^T_{i+1} H \dot{q}_{i+1} - \dot{q}^T_{i+1} g$,
8. calculer la nouvelle consigne articulaire $q^e_{i+1} = q_i + \dot{q}^e_{i+1}.t_{cycle}$.

Ce chapitre présente ma contribution à l'extension de la bibliothèque *rtrmac* par un choix de formalisation du problème d'entraînement multiaxe. Cette formalisation est basée sur une description générique des architectures de mécanismes et de robot qui permet de traiter d'une façon homogène des systèmes très différents. Cette méthode, illustrée sur une architecture simple, est construite pour pouvoir être mise à l'échelle vers des systèmes avec des nombres d'axes plus importants.

La séparation entre les fonctions de génération de trajectoire et le calcul des consignes articulaires permet d'envisager d'autres stratégies d'optimisation par programmation quadratique, comme par exemple les approches basées sur la surveillance et la limitation de l'énergie cinétique des pièces en mouvement, cf. [Joseph 2020]. Les fonctions présentes dans la bibliothèque sont prévues pour être implémentées dans des calculateurs industriels conformes au standard IEC61131. Le logiciel de contrôle commande est exécuté dans un système d'exploitation temps-réel, au plus près des préactionneurs et des modules de conditionnement des capteurs. Cependant, les contraintes retenues sur le langage de programmation et l'intégration de bibliothèques tierces permettent d'étendre l'usage de notre approche à des cartes électroniques embarquées et à des simulateurs associés à des jumeaux numériques. Ces différents aspects sont présentés dans le dernier chapitre.

Chapitre 4

Du jumeau numérique aux cartes embarquées

Ce chapitre présente plusieurs exemples qui illustrent la diversité des cas d'usages possibles de notre démarche de développement de logiciel de contrôle-commande. J'aborde tout d'abord la stratégie choisie par l'équipe pour le développement de jumeaux numériques connectés. Ces outils logiciels permettent de tester et valider le fonctionnement de systèmes mécatroniques en avance de phase sur la réalisation du dispositif physique, puis permettent à l'opérateur de contrôler le système en phase de production et peuvent également apporter une assistance à son maintien en condition opérationnelle.

Le principe de création d'un jumeau numérique connecté est, dans un premier temps, illustré par l'exemple d'un démonstrateur robotique simple associant un mode de simulation et un mode de conduite d'une maquette réelle. Ces deux situations utilisent la même version du logiciel de contrôle-commande.

La même approche est retenue pour la réalisation du logiciel de pilotage d'un préhenseur robotique sous-actionné. Dans un premier temps, la démarche de création des fonctions spécifiques à un nouveau robot est illustrée. Le développement du logiciel du prototype et son implémentation sur une architecture matérielle à deux calculateurs sont ensuite détaillés.

4.1 Conception de jumeaux numériques fidèles

La complexité croissante des machines, des dispositifs mécatroniques à concevoir, la nécessité de pouvoir simuler la gestion temporelle stricte, de reproduire fidèlement en simulation le comportement de la machine réelle, ont fait évoluer le besoin de simulation des systèmes cyber-physiques. C'est ce qui a fait émerger le concept de jumeau numérique connecté qui est élaboré dans ce chapitre.

4.1.1 Jumeau numérique : concept et définitions

La combinaison des technologies de l'automatisation et de la virtualisation doit contribuer à répondre aux défis de production de l'usine du futur par le développement de nouvelles solutions d'assistance et d'aide à la décision. Ces solutions pourraient permettre aux acteurs humains de valoriser leurs savoir-faire et de gagner en productivité et en confort pour répondre aux enjeux majeurs d'une production flexible et personnalisée. Ceci a conduit l'équipe RoBioSS à amorcer en 2017 une collaboration avec l'entreprise ITECA pour produire une première preuve de concept de jumeau numérique connecté entre une cellule de production robotique réelle et sa maquette virtuelle. En 2019, le laboratoire commun ANR MACH4 avec ITECA a été créé pour accélérer la production de jumeaux numériques connectés afin de relever ces défis.

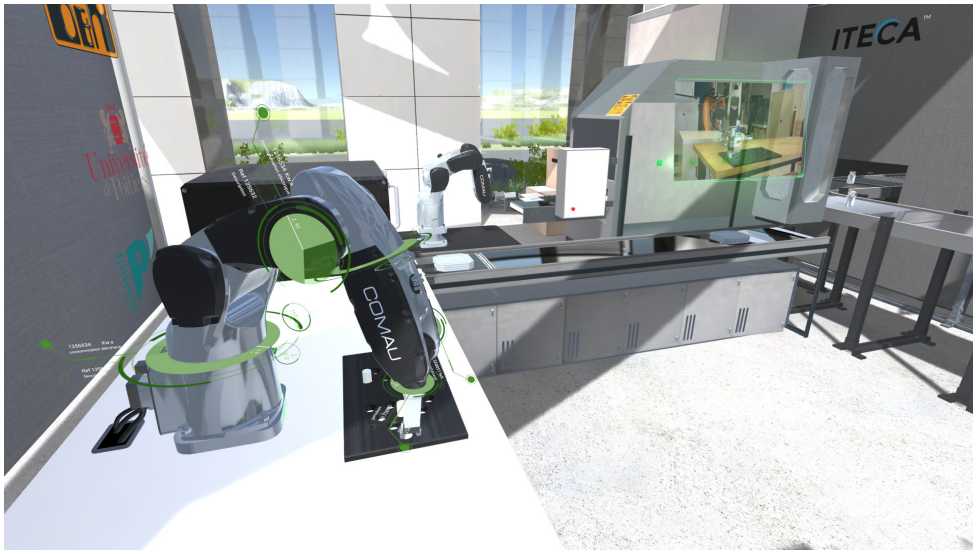


FIGURE 4.1 – Scène virtuelle d'une cellule de production © ITECA.

La figure 4.1 illustre une scène virtuelle d'une cellule de production industrielle. Cette maquette 3D qui représente la géométrie des pièces de la cellule ne constitue qu'une partie du jumeau numérique de l'installation.

Plusieurs définitions sont proposées dans la littérature mais elles sont souvent dépendantes des domaines d'application. Les principaux domaines sont décrits ci-dessous.

L'industrie de la construction et la ville intelligente L'adoption généralisée de la modélisation des informations du bâtiment (Building Information Modeling ou BIM) et l'émergence récente des applications de l'Internet des objets (IoT) offrent de nouvelles perspectives pour accompagner la prise de décision tout au long du cycle de vie des bâtiments. L'industrie de construction met en œuvre plusieurs stratégies pour accompagner cette prise de décision et réduire son impact carbone et le jumeau numérique des ouvrages dans le bâtiment constitue l'un de ses leviers stratégiques. Le jumeau numérique associe dans ce contexte : maquette numérique 3D et base de données associée, les liens et interactions entre les objets physiques et numériques, l'analyse comportementale de l'ouvrage soumis à diverses sollicitations (mécaniques, thermiques, énergétiques, etc.) [Deng 2021]. A titre d'exemple, EDF utilise la simulation numérique depuis des années pour prédire et analyser le comportement de ses structures et de ses ouvrages de production d'énergie en fonction des données collectées que ce soit sur le terrain ou via des campagnes d'essais spécifiques. Le développement récent des méthodes d'analyse des données basées IA et des techniques de réduction de modèle couplées avec des approches « data-driven » permettent d'envisager des modèles hybrides plus fidèles à la réalité couplant essais et calculs numériques. Au-delà de la construction, le jumeau numérique de la ville intelligente peut faciliter sa croissance, accompagner son évolution en testant des scénarios et en apprenant en permanence de l'environnement via la collecte des données capteurs et des matériels connectés [Soe 2017], [Chen 2018].

L'énergie La numérisation des industries de transformation de l'énergie grâce à la technologie des jumeaux numériques promet des améliorations progressives sur plusieurs aspects :

— la gestion et l'optimisation de l'énergie,
— l'amélioration de l'entretien et de la maintenance,
— la conception des nouveaux sites et l'amélioration des sites de production existants [Morilhat 2018], et
— l'intégration des énergies renouvelables produites localement et régionalement [Yu 2022].

Le secteur de la production d'énergie nucléaire est certainement l'un de ceux pour lesquels les jumeaux numériques sont les plus élaborés. Les enjeux de la modélisation de systèmes complexes, de la taille et de la difficulté d'accès des installations, de conduite et de maintenance d'installation critique et de la formation des différents opérateurs sont tous présents. La figure 4.2 présente les différentes aspects à prendre en considération pour la réalisation du jumeau numérique d'une centrale nucléaire [Yadav 2021].

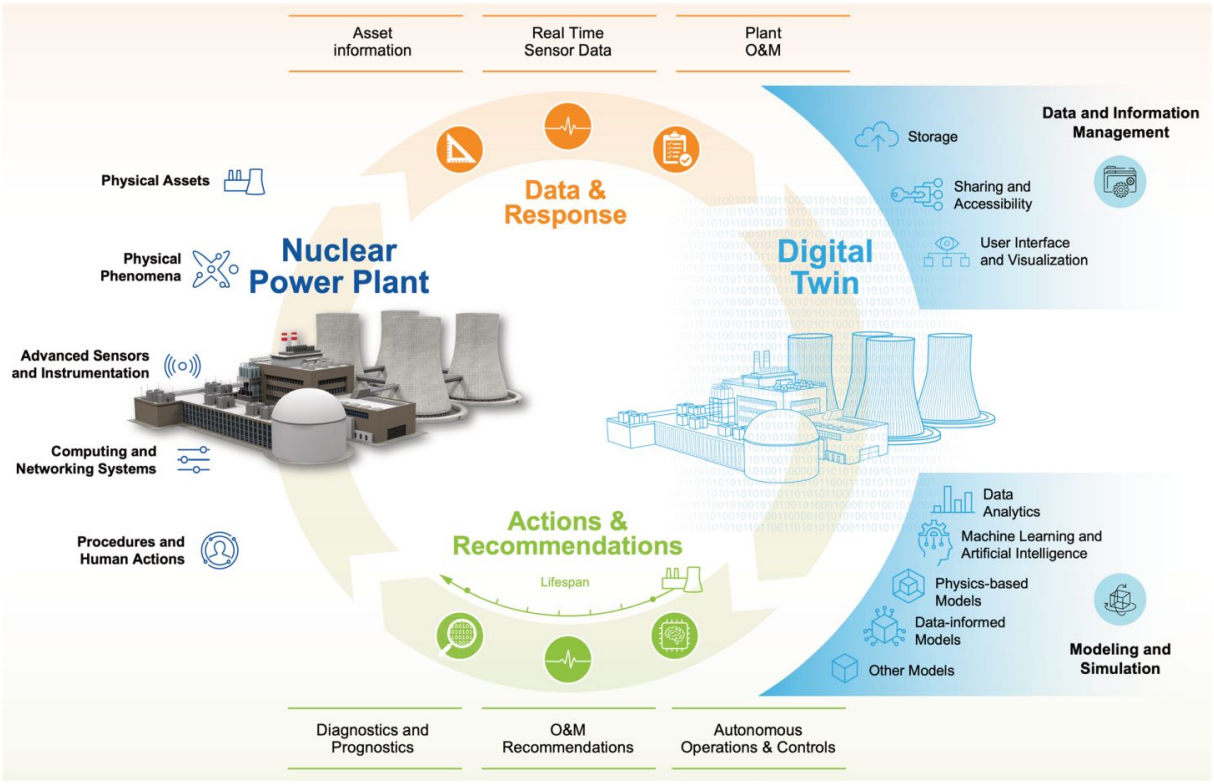


FIGURE 4.2 – Conception du jumeau numérique d'une centrale nucléaire, extrait de [Yadav 2021].

Le monde médical La modélisation personnalisée du corps humain est un enjeu majeur dans le domaine médical, la modélisation fine de l'individu à partir de ses propres données de santé (obtenues à partir de techniques d'imagerie par exemple) offre la possibilité de créer son double virtuel. L'exploitation de cette représentation numérique permet d'accompagner le praticien pour envisager des prédictions et recommandations personnalisées pour le patient ou d'accompagner

le chirurgien dans la préparation d'interventions chirurgicales complexes, l'orthopédiste dans la production de prothèses adaptées. Les jumeaux numériques de services de santé, tels que rapportés dans la littérature[Bertezene 2022], permettent quant à eux de modéliser des systèmes complexes tels que des services d'urgence, de soins intensifs, de radiologie à partir des données qui sont collectées (nombre de patients sur une période donnée, examens, actes médicaux dans un service par type et par heure, etc.). Les objectifs poursuivis permettent de gérer plus efficacement les scénarios et les ressources (médicaments, disponibilité des équipements et fournitures, temps, personnels disponibles, flux de patients) en raison des aléas et des difficultés identifiées. Un jumeau numérique a été développé par Siemens Healthineers [GmbH 2018] afin d'améliorer les processus hospitaliers au sein du département radiologie de l'hôpital Mater Private Hospitals en Irlande et pour faire face à l'augmentation du nombre de patients et des temps d'attente associés. Une animation 3D réaliste y est associée à la collecte des données pour simuler les scénarios de fonctionnement du service.

L'industrie et le contrôle de process Le jumeau numérique est au cœur des enjeux propres à l'usine du futur. Il accompagne l'acquisition, l'échange et le partage de quantités de données variées contextualisées, accessibles au bon moment et au bon endroit constituant ainsi un outil majeur de la transformation numérique de l'industrie. Dans [Gregorio 2020], l'auteur rappelle que l'industrie aéronautique et spatiale, a été l'une des premières industries à exploiter le concept de jumeau numérique pour simuler numériquement le comportement mécanique des structures des aéronefs. Au-delà des défis posés par le développement de produits complexes, le jumeau numérique permet de superviser et gérer les performances de produits en fonctionnement afin d'en améliorer la sécurité, les performances et la satisfaction des clients. L'entreprise GE a ainsi créé une plateforme nommée « Predix » [Fuller 2020] utilisée pour l'analyse et la surveillance des données. De même Siemens a développé une plateforme appelée « MindSphere » avec un système basé sur le cloud qui connecte les machines et l'infrastructure physique afin d'accompagner la transformation de l'industrie par la production de jumeaux numériques basés sur les données et sur leur exploitation. A titre d'exemple, la figure 4.3, montre le jumeau numérique d'un robot connecté au produit réel et exploité avec un casque de réalité augmentée.

La gestion de production Dans le contexte de la production de biens manufacturés, le jumeau numérique a le potentiel de fournir une information continuellement actualisée sur les performances des machines et de la ligne de production [He 2019], [Mawson 2019]. La virtualisation de la production offre ainsi la possibilité de simuler et évaluer les différentes stratégies de fabrication et de planification. Les modèles virtuels se mettent à jour en fonction des données collectées provenant du monde physique, ce qui permet d'identifier et optimiser par la simulation dans le jumeau numérique le processus de production [Qi 2018].

Le concept de jumeau numérique est relativement récent puisqu'il a été introduit dans un livre blanc par Grieves en 2003 [Grieves 2014].

Les tableaux 4.1 4.2, d'après [Harper 2019] présentent les principales caractéristiques des jumeaux numériques et les liens avec les cas d'utilisation.

Plusieurs définitions sont proposées dans la littérature, une sélection d'entre elles sont introduites ci-dessous en respectant la chronologie des publications associées :

— La NASA définit le jumeau numérique comme étant une simulation multiphysique, multiéchelle et probabiliste intégrée d'un véhicule ou d'un système tel qu'il a été construit,

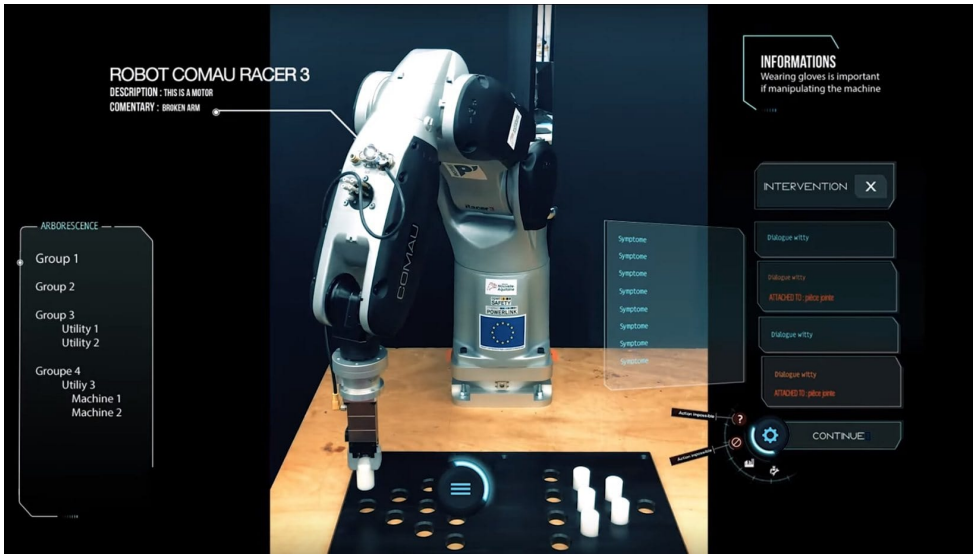


FIGURE 4.3 – Enrichissement d'une cellule robotique par ajout d'informations en réalité augmentée (© ITECA).

TABLE 4.1 – Caractéristiques des jumeaux numériques [Harper 2019].

Caractéristique	Définition
Gestion documentaire	Archivage et suivi d'évolution de tous les documents associés à un équipement tout au long de son cycle de vie
Modèle	Représentation numérique de l'équipement qui contient ses propriétés et ses comportements
Représentation 3D	Propriétés d'un dispositif (mesurées ou simulées) mises en correspondance avec une représentation numérique en 3D
Simulation	Représentation d'un dispositif dans un environnement de simulation afin d'étudier son comportement
Modèle de données	Modèle de données normalisé pour la connectivité, l'analyse et/ou la visualisation
Visualisation	Représentation graphique de l'objet sur un périphérique de supervision
Synchronisation	Actualisation périodique du modèle sur des informations du monde réel
Analyses statistiques	Algorithmes et résultats de calcul basés sur les propriétés mesurées du dispositif

qui utilise les meilleurs modèles physiques disponibles, les mises à jour des capteurs, l'historique de la flotte, etc. pour reproduire la vie de son jumeau « réel » [Glaessgen 2012].

— Le jumeau numérique peut être défini comme une représentation virtuelle dynamique d'un objet physique ou d'un système tout au long de son cycle de vie, utilisant des données

Table 4.2 – Liens avec les cas d'utilisation [Harper 2019].

	Conception	Industrialisation	Exploitation	Maintenance
Gestion documentaire	PLM	PLM	Instructions de conduite	Suivi modifications
Modèle	Prédiction des propriétés physiques		Optimisation	Diagnostic
Représentation 3D	Définition des composants et des ensembles	Instructions		Instructions
Simulation	Simulation pour la conception	Mise en service virtuelle		
Modèle de données	Données d'ingénierie	Exigences de production	Données opérationnelles	Retour d'expérience
Visualisation			Etat de fonctionnement	Suivi des indicateurs de maintenance
Synchronisation			Actualisation en continu	Aide à la maintenance corrective
Analyses statistiques			Indicateurs de performance	Suivi de l'état de l'installation

qui s'actualisent pour permettre la compréhension de son évolution [Bolton 2018].

— Un jumeau numérique est une représentation numérique d'un article ou d'un assemblage physique utilisant des simulations intégrées et des données de service. La représentation numérique contient des informations provenant de sources multiples tout au long du cycle de vie du produit. Ces informations sont mises à jour en permanence et sont visualisées de diverses manières pour prédire le comportement actuel et futur, tant dans les environnements de conception que dans les environnements opérationnels, afin d'améliorer la prise de décision [Vrabič 2018], [Liu 2018].

— Une dernière définition considère le jumeau numérique comme une représentation numérique d'un produit unique et actif (appareil réel, objet, machine, service ou actif incorporel) ou d'un système produit-service unique (un système composé d'un produit et d'un service associé) qui comprend ses caractéristiques, propriétés, conditions et comportements au moyen de modèles, d'informations et de données au sein d'une seule voire de plusieurs phases du cycle de vie [Stark 2019].

A partir de ces différentes approches, il est possible de converger vers les points synthétiques

suivants :

— Le jumeau numérique est une représentation numérique d'un système physique, constituée d'un ensemble de modèles du système, mise à jour en continu avec les données générées, acquises, stockées et consommées tout au long du cycle de vie du système physique ;
— Comme évoqué dans [Fuller 2020], si les données circulent dans les deux sens entre un objet physique existant et sa représentation numérique, il s'agit d'un jumeau numérique. Une modification apportée à l'objet physique entraîne automatiquement une modification de sa représentation numérique et vice versa.

4.1.2 Particularisation du jumeau numérique pour les machines de production

Dans le domaine de l'industrie et particulièrement de l'automation et du contrôle multimachine ou multirobot dans l'environnement de production, on conçoit que le jumeau numérique permet d'adresser au niveau applicatif, en temps réel, la supervision de processus, son optimisation, sa maintenance, son évolution ainsi que la formation des opérateurs. La connexion continue du système réel avec son jumeau numérique permet ce mimétisme permanent entre les environnements virtuel et réel.

Si on reprend les types de jumeaux numériques proposés dans [Syntec-Ingénierie 2021] illustrés dans le tableau 4.3, on conçoit que seul le jumeau numérique connecté avec ses homologues, c'est-à-dire le jumeau numérique interconnecté offre le niveau d'interaction attendu pour répondre aux enjeux visés de l'usine du futur.

Il s'agit ainsi de proposer dans ce contexte, un jumeau numérique interconnecté capable de se connecter à la fois aux autres jumeaux numériques, mais également aux applications à travers les APIs (Application Programming Interfaces). Le succès de cette proposition de jumeau est fortement lié aux caractéristiques du jumeau numérique afin d'accompagner la totalité du cycle de vie du système physique depuis l'écriture du cahier des charges, jusqu'à la conception, au fonctionnement opérationnel et à la maintenance comme illustré sur la figure 5. Dans [Malakuti 2019], les auteurs montrent à travers l'exemple de l'industriel ABB la nécessité d'appuyer les caractéristiques des jumeaux numériques sur des standards pour répondre aux enjeux de consultation et de modification des bases de données et de connexion sécurisée au jumeau numérique par des logiciels clients différents.

Dans le contexte du soutien à l'automation, à la robotisation dans l'usine du futur, nous proposons une démarche de développement des jumeaux numériques basée sur des standards d'automatisation entre la machine simulée et la machine réelle. Cette stratégie offre le double avantage

— de pouvoir tester le comportement de la machine en amont,
— de pouvoir piloter de manière transparente soit la machine virtuelle à des fins de simulation ou de formation, soit la machine réelle en mode de contrôle supervisé, en mode de contrôle à distance ou en mode virtuel et réel simultanément.

Notre définition de la *fidélité de simulation* est relative à la reproduction du comportement cinématique et dynamique de la machine réelle dans l'environnement de simulation en s'appuyant sur le déterminisme temporel aussi bien du côté du système physique, que de sa représentation virtualisée.

Table 4.3 – Les différents types de jumeaux numériques [Syntec-Ingénierie 2021].

	Type de jumeau	Contenu du modèle numérique	Niveau d'échange des données	Utilisation, fonctionnalités
4	Interconnecté	Interconnexion de plusieurs modèles numériques formant un ensemble fonctionnel cohérent	Échange entre différents jumeaux. Ensemble d'infrastructures.	Permet d'organiser un fonctionnement connecté et adapté pour un ensemble d'ouvrages, d'infrastructures ou d'équipements (smart city)
3	Communicant	Modèle numérique avec intégration des données acquises en temps réel (capteurs, IoT, actionneurs...)	Échanges bidirectionnels entre jumeaux numérique et réel (smart building, usine 4.0)	Permet de lier les données dynamiques ou temps réel d'un process pour le piloter, voire associer un algorithme qui réalise le pilotage du process
2	Dynamique	Modèle numérique avec intégration des données acquises en temps réel (capteurs, IoT...)	Interface entre les systèmes opérants, connexion unidirectionnelle	Extraction des données dynamiques et exploitation des données de fonctionnement (maintenance conditionnelle)
1	Statique	Modèle numérique d'objets avec informations non géométriques	Pas d'échange	Extraction de données d'entrée statiques pour conception, calcul et simulation

4.2 Démonstrateur numérique pour l'éducation

Cette section présente la réalisation d'une petite machine de production qui est associée à un jumeau numérique, comme illustré fig. 4.4. La totalité du développement matériel et logiciel a été réalisée par l'équipe Pprime dans le cadre des activités du Labcom ANR Mach4.

Il s'agit d'un robot cartésien deux axes dont l'effecteur évolue dans le plan vertical. Ce système est une version miniature de portiques cartésiens industriels, il nous sert à évaluer différentes modalités d'interaction entre un opérateur et un système automatisé en réalité virtuelle et est également utilisé comme support de travaux pratiques, en enseignement d'automation à

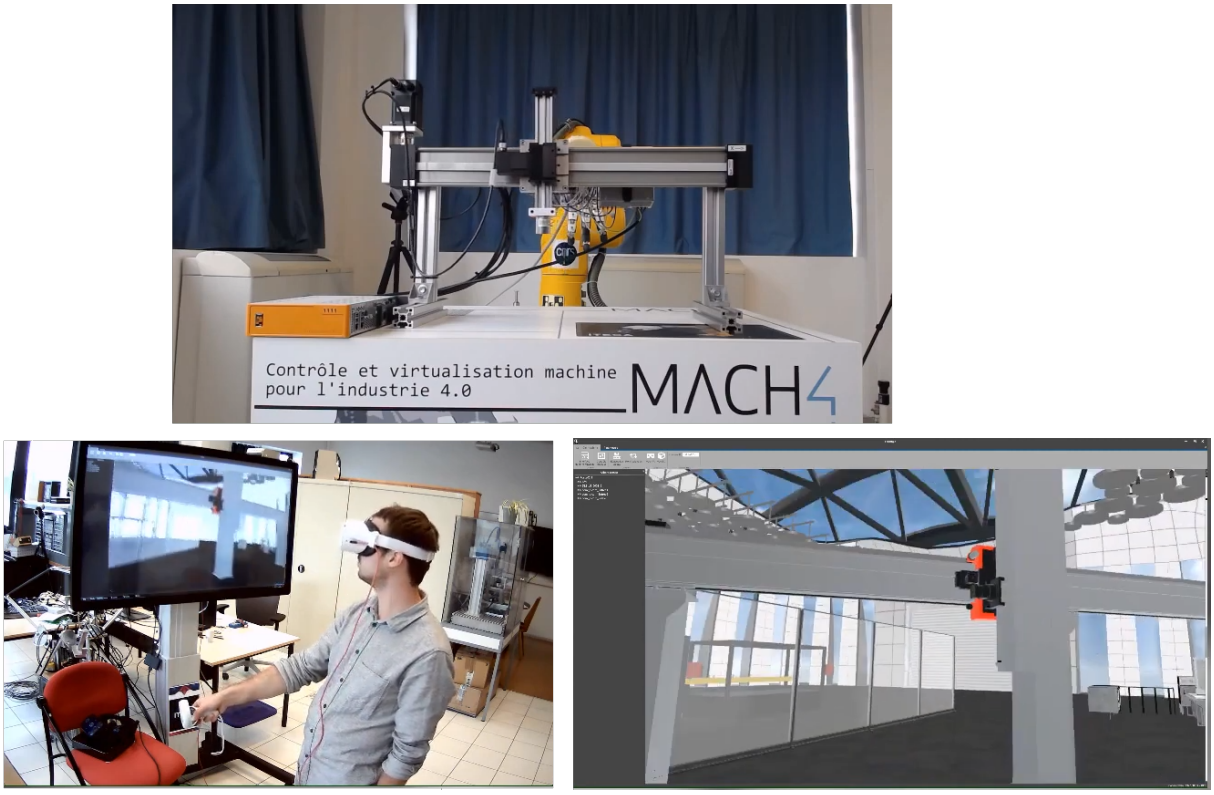


FIGURE 4.4 – Dispositif Mach4 Edlab : robot réel, usage et visualisation du jumeau numérique.

l'université de Poitiers. Ce dispositif a été mis au point en collaboration avec ITECA et a été lauréat en 2021 de l'appel à projet *EdLab* de la région Nouvelle Aquitaine « Expérimentations de solutions numériques innovantes pour l'éducation et la formation professionnelle ». [1]

Ce robot est intégré dans une cellule de production sous la forme d'une machine de transfert automatisée avec le choix de scénarios de transfert de pièces entre convoyeurs ou de palettisation. Ces mises en situation représentent deux tâches industrielles courantes. Cette machine permet d'illustrer le concept de jumeau numérique connecté défini dans la section précédente. En effet, le même jumeau numérique peut être utilisé :

— en phase amont de la conception d'une machine pour tester et valider le logiciel de pilotage, évaluer a priori les performances de l'installation par l'estimation des cadences de production en fonctionnement nominal ou en mode dégradé ou pour former un opérateur,
— en conduite de ligne et en maintien en condition opérationnelle, comme interface utilisateur pour un conducteur de ligne ou un technicien de maintenance.

Dans ce second cas, la visualisation du jumeau numérique présente l'interface homme-machine qui s'affiche sur les écrans tactiles de conduite de la machine réelle. Ce logiciel est exécuté par l'automate programmable et mis à disposition par un protocole de prise en main à distance. Le serveur est exécuté par l'automate, le client par le périphérique distant (PC de supervision ou casque de réalité virtuelle).

1. Le principe de ce dispositif alliant maquette physique et jumeau numérique est présenté en vidéo sur src/koda.cnrs.fr.

Cette IHM donne accès aux modes de fonctionnement implémentés dans le programme de gestion machine : modes de production automatique ou semi-automatique et mode manuel, par exemple. Un mode complémentaire permet de téléopérer la maquette à partir d'interfaces mises à disposition par la scène 3D du jumeau numériques. Ce sont des éléments d'interaction que l'opérateur peut manipuler à la souris, sur un écran tactile ou par les manettes des casques de RV, selon le périphérique choisi.

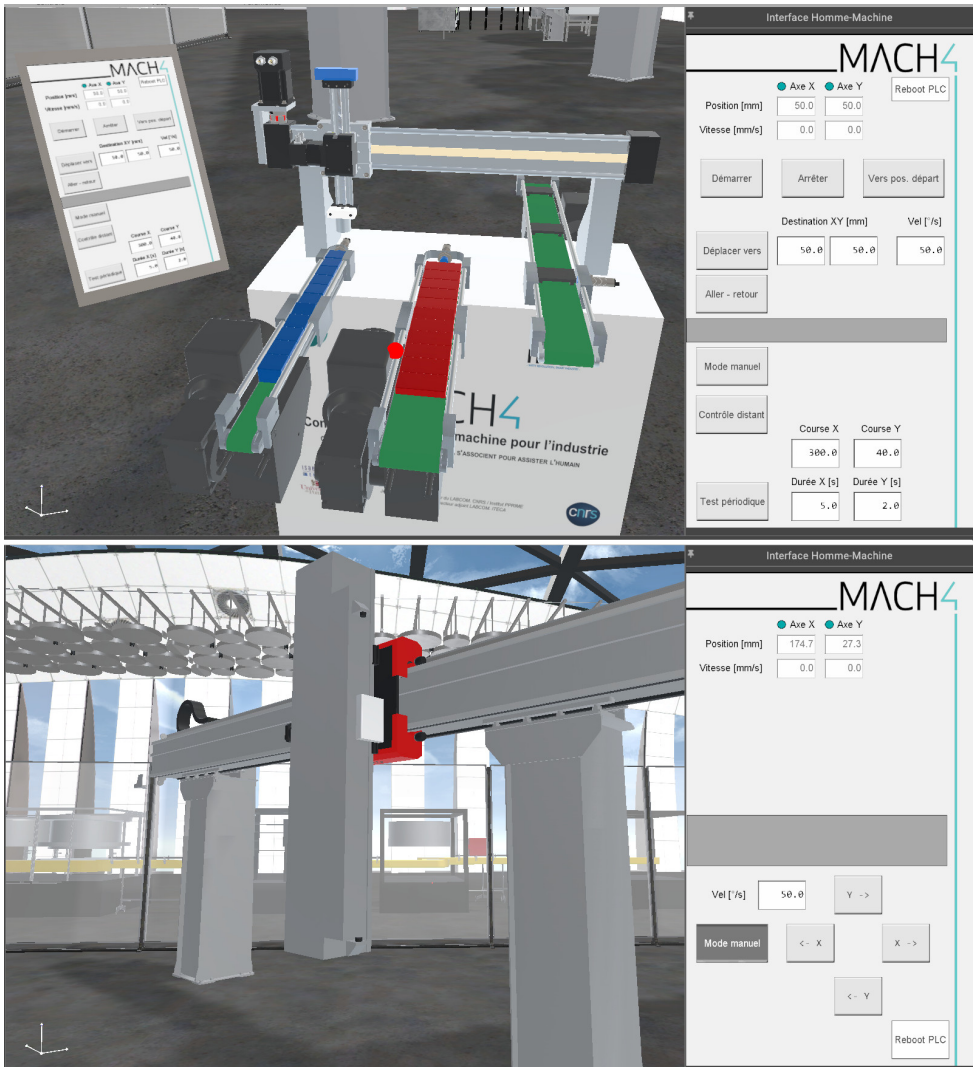


FIGURE 4.5 – Exemples de cas d'usage d'un jumeau numérique : simulation de l'environnement de la partie opérative (haut) ou mise à disposition virtuelle d'une cellule de grandes dimensions (bas).

Dans le cas de l'usage de cet équipement pour des activités de formation, le jumeau numérique présente également d'autres aspects intéressants. Son premier usage est de mettre en scène le portique cartésien et de simuler des interactions avec d'autres éléments de la ligne de production. Il permet également de généraliser et d'abstraire le travail de génération de cycle de fonctionnement et de définition de trajectoire pour l'appliquer à des architectures similaires mais

de plus grande taille et de plus grande dynamique qui ne sont pas financièrement et matériellement accessibles pour des centres de formation (cf. figure 4.5). Enfin, sur des aspects purement éducatifs, un jumeau numérique permet de dupliquer une installation pour faire travailler des groupes en parallèle avant la phase de déploiement du logiciel sur l'installation réelle. L'ajout d'options de paramétrisation de la scène virtualisée et l'intégration dans le jumeau numérique de métriques mesurant la progression des étudiants en situation de développement du logiciel machine permettent d'envisager une personnalisation du travail à réaliser et de l'évaluation des compétences à acquérir.

Les deux modes de fonctionnement du dispositif sont présentés sur la figure 4.6 :

— il peut être utilisé en version *développement* avec le jumeau numérique en outil de visualisation, de diagnostic et d'auto-évaluation, puisque cet outil permet de valider les trajectoires programmées et la réalisation des fonctions de transfert ou de palettisation,

— lorsque les programmes sont validés en simulation, ils sont portés sur le système réel. Ce second mode de fonctionnement permet également d'évaluer plusieurs modes d'interaction entre l'utilisateur et le système automatisé.

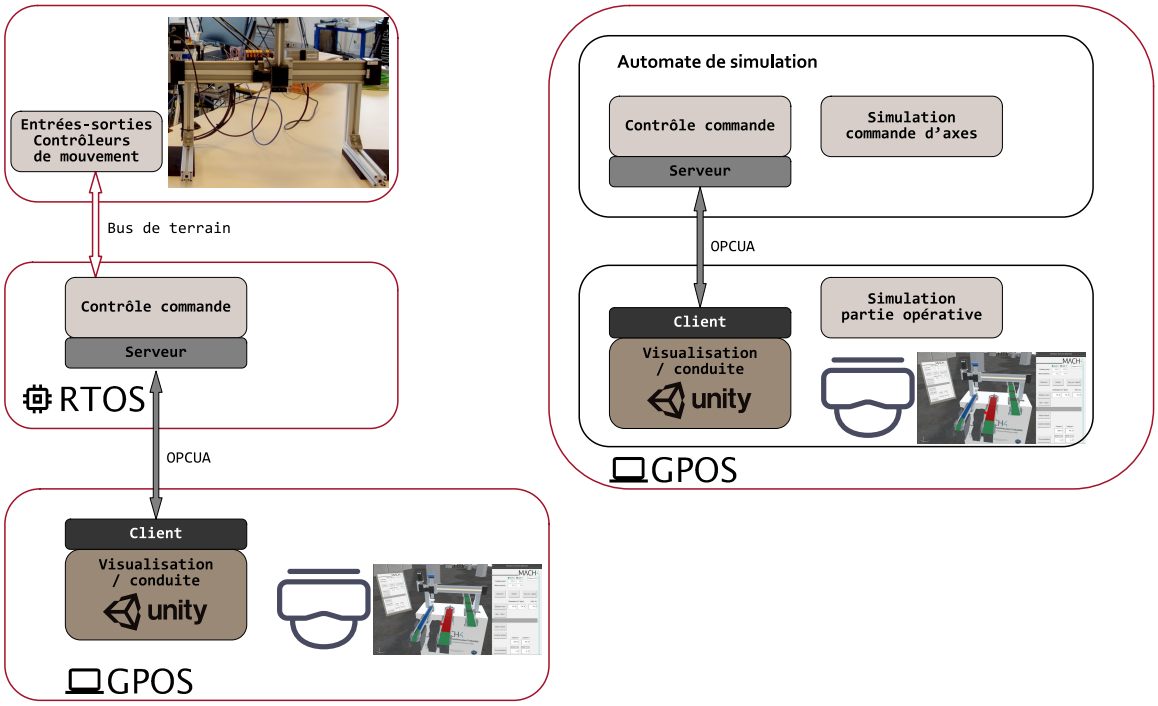


FIGURE 4.6 – Architectures logicielles du dispositif en configuration de conduite (gauche) et en développement du contrôle-commande (droite).

Le découpage logiciel du contrôle-commande est structuré en plusieurs couches qui correspondent à des fonctions techniques distinctes et à des niveaux de compétences différents (cf. figure 4.7). Cette structure hiérarchique est une version simplifiée de la structure générique implémentée dans la bibliothèque *rtrmac* présentée au chapitre 3.2.2.

Selon le niveau d'étude et l'objectif pédagogique choisi, les étudiants ont à réaliser ou à com-

pléter le programme correspond à l'une des couches fonctionnelles de la figure 4.7. Les premières applications portent sur la réalisation du programme de gestion machine ou sur la modification de l'interface homme-machine. Les étudiants des formations en automation pourront également produire un bloc fonctionnel de gestion, travailler sur la coordination d'axes et sur le paramétrage des interfaces matérielles.

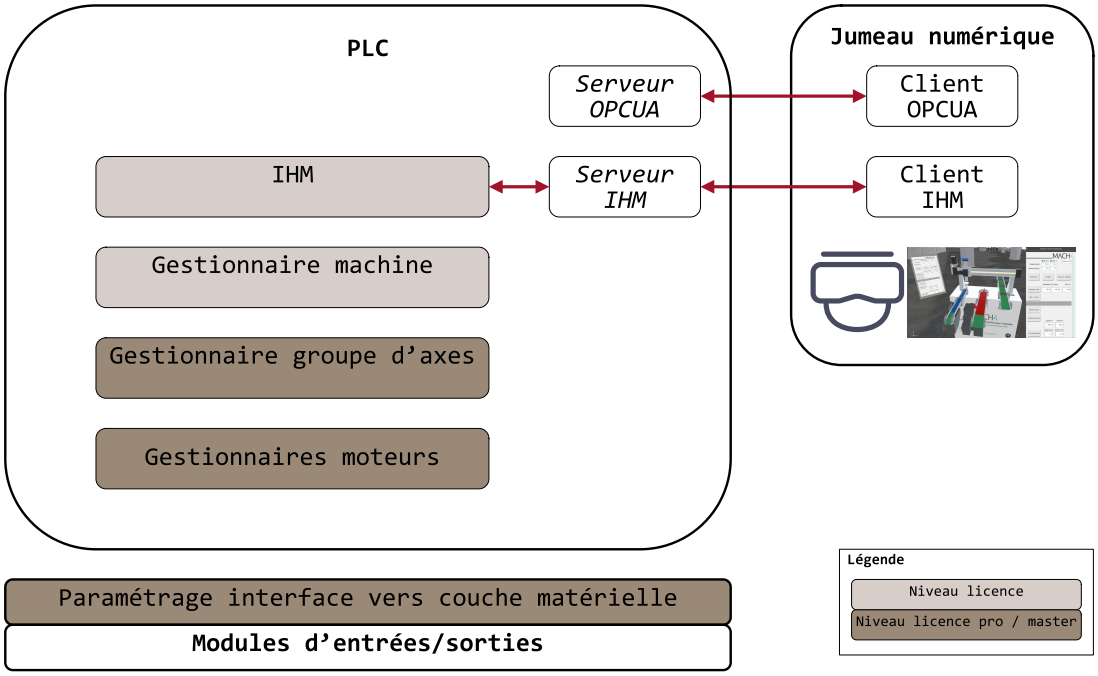


FIGURE 4.7 – Décomposition fonctionnelle du projet de base du dispositif Edlab.

Le système est en production (en situation d'assistance à la formation) depuis deux ans en licence professionnelle et en licence sciences pour l'ingénieur à l'université de Poitiers. L'activité de conception de dispositifs numériques pour la formation se poursuit dans le cadre du projet DemUP, lauréat de l'appel à projets PIA 4 AMI DEMOES « Démonstrateurs numériques pour l'enseignement supérieur » et qui a commencé en janvier 2022.

Les développements en cours portent sur l'ajout de dispositifs permettant de mesurer la progression des étudiants et la contribution de ces dispositifs d'assistance à l'acquisition de compétences professionnelles. Un second système automatisé avec un jumeau numérique associé est en cours de réalisation, avec un objectif d'intégration comme support d'enseignement de robotique et de dynamique des solides.

4.3 Création du modèle robotique d'un robot sous actionné

4.3.1 Présentation du préhenseur Seahand

Le projet Seahand, financé par l'Agence Nationale de la Recherche, portait sur la conception et la réalisation d'une main robotisée pour saisir des objets archéologiques fragiles en eaux pro-

fondes. Il a associé quatre partenaires : l'Institut Pprime, le LIRMM, le DRASSM (Département des recherches Archéologiques Subaquatiques et Sous-Marines), et le bureau d'étude Becom-d (Saint-Jean-de-la-Blaquière, Hérault).

Dans ce cadre, les travaux de thèse de Camille Mizera [Mizera 2019] ont porté, pour partie, sur la conception d'une architecture robotique permettant de réaliser un grand nombre de tâches opérées par un plongeur archéologue lors de fouilles sous-marines. La figure 4.8 présente quelques exemples de tâches de saisie nécessaires au plongeur qui peuvent être reproduites par le préhenseur robotique.

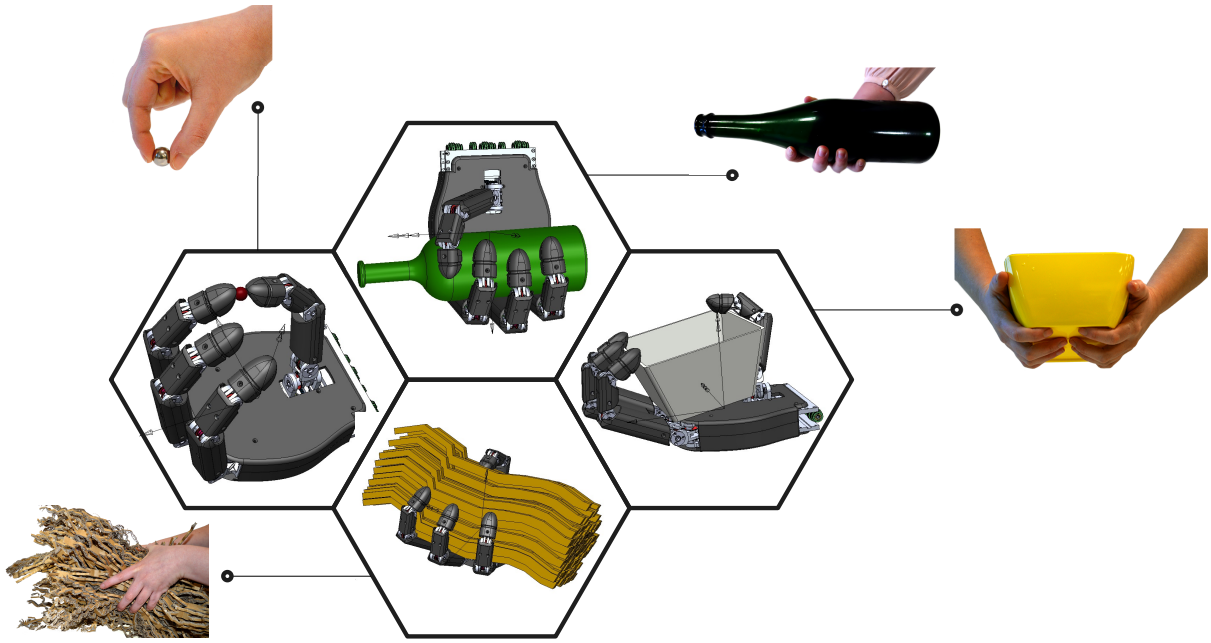


FIGURE 4.8 – Exemples de prises réalisables avec l'architecture choisie.

Les contraintes de conception étaient nombreuses : le préhenseur doit être déplacé par un véhicule sous-marin téléopéré, la puissance électrique pour les actionneurs est limitée, l'électronique de commande doit être embarquée, la liaison avec le système de pilotage sur le bateau a une bande passante limitée et le préhenseur doit pouvoir évaluer et transmettre à l'opérateur une estimation des efforts d'interaction. Les exigences environnementales d'un travail en mer en grande profondeur sont évidemment très contraignantes.

Le travail de conception optimale a conduit au design d'une main à quatre doigts (trois doigts longs et un pouce en opposition), avec quatre articulations par doigt. Ce mécanisme est fortement sous actionné avec six moteurs qui entraînent quinze articulations. Ce préhenseur est présenté sur la figure 4.9. Les six motoréducteurs sont placés dans la partie arrière de l'avant-bras et les transmissions entre les poulies motrices et les poulies qui entraînent les articulations sont réalisée par des câbles textiles polymères.

Le travail de conception de l'électronique de commande a été réalisé en partenariat avec le Centre de Ressources Technologiques CATIE (Talence, Gironde). La solution réalisée est basée sur un microcontrôleur STM32 qui exécute le logiciel de contrôle-commande bas niveau et communique avec un contrôleur industriel. L'étage de puissance permettant le pilotage des

moteurs à courant continu est réalisé par des modules Escon qui fournissent une régulation courant - vitesse - position.

La partie suivante décrit le travail réalisé pour le développement du logiciel de contrôle-commande de la main et se sépare en trois parties :

— la modélisation du robot et la création des objets logiciels programmables spécifiques pour le contrôle de ce système et leur validation en simulation,
— le développement du logiciel de pilotage d'un banc de test permettant de qualifier les performances d'un doigt,
— le développement d'une première version du logiciel de pilotage du préhenseur.

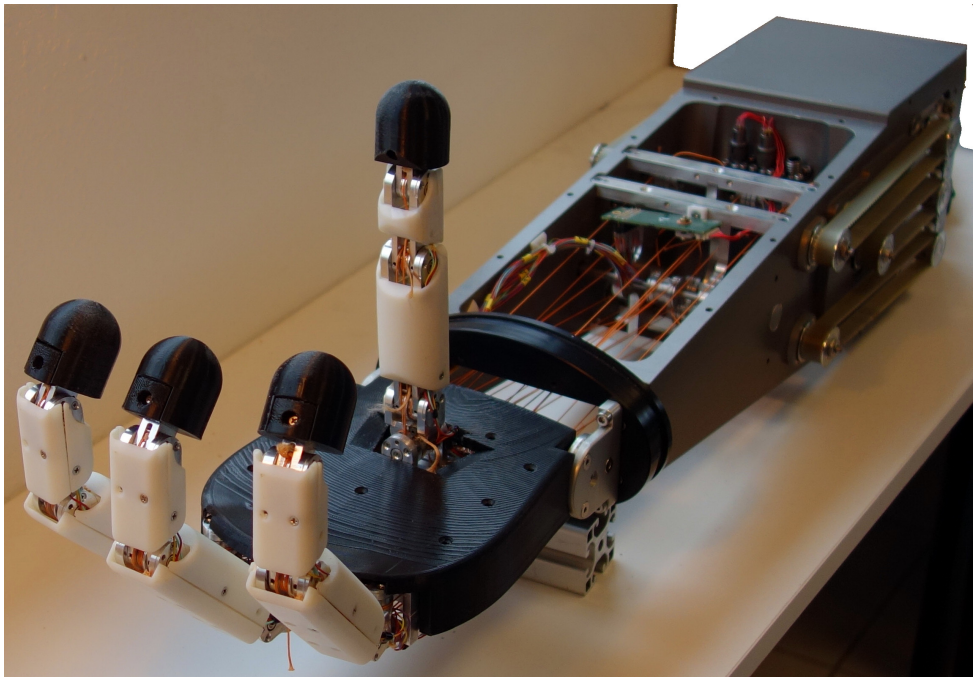


FIGURE 4.9 – Préhenseur Seahand.

4.3.2 Description du système

La main Seahand constitue un bon exemple de mise en œuvre de la démarche de description de robot présentée précédemment pour plusieurs raisons. Ce mécanisme possède une architecture complexe avec un fort sous-actionnement et les effecteurs étant fixes sur l'avant-bras, les chaînes de transmission présentent des couplages. Enfin, le choix d'utiliser des transmissions par câbles permet aux doigts de s'adapter à l'objet à attraper, ce qui augmente les possibilités de saisie et permet également d'envisager d'évaluer les actions de saisie. Il sera par contre nécessaire de modéliser le comportement de ces transmissions de mouvement déformables pour les intégrer dans la commande.

Ce préhenseur robotique constitue également un exemple difficile à traiter de système multi-robot puisque le pouce et le groupe constitué des trois doigts longs peuvent être décrits comme deux robots dont la coordination est imposée par la tâche haut niveau de saisie de l'objet.

Les mouvements en sortie des motoréducteurs sont transmis par des câbles à des poulies liées

aux phalanges des doigts. Ces liaisons sont bidirectionnelles : deux câbles entre poulies motrices et réceptrices permettent le mouvement et la transmission d'effort dans les articulations dans les deux sens de rotation. Tous les motoréducteurs sont fixés sur le bâti de la main, la complexité de la conception de la main Seahand portait, en partie, sur la réalisation d'un routage des câbles qui permettent de découpler le mouvement d'abduction des doigts des mouvements de flexions entre phalanges.

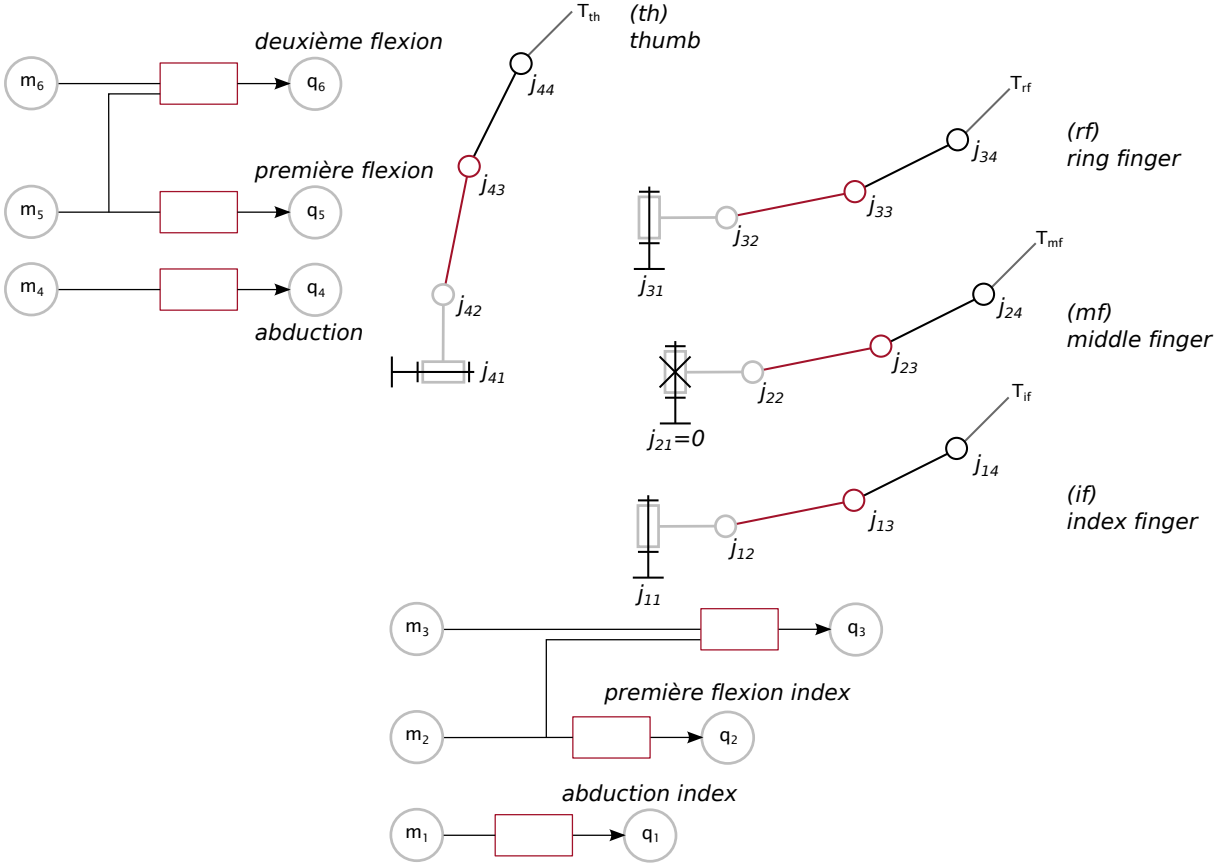


FIGURE 4.10 – Seahand : paramétrage des moteurs et des articulations.

Modélisation de la partie *mécanisme* La définition des modèles de transmission et des modèles robotiques de ce système fait l'objet de l'annexe B.

La partie *mécanisme* de notre système décrit le passage, via des transmissions poulies-courroies et les transmissions par câbles des positions moteurs à un nombre équivalent de positions articulations qui vont définir la configuration de l'ensemble du mécanisme. Ces six positions articulaires correspondent aux coordonnées *ACS* (axis coordinate system), définies dans le chapitre précédent. Les articulations retenues sont les abductions-adductions, les premières et deuxièmes flexions d'un doigt long (l'index, si le préhenseur est considéré comme une main droite) et celles du pouce. La figure B.1 de l'annexe B présente les relations qui vont définir les relations de couplage entre les positions moteurs et les positions ACS. Dans la suite de ce manuscrit, ces relations de couplage sont définies comme des relations linéaires, mais l'étude du

comportement visco-élastique des câbles est prévue dans les prochains mois. Les fonctions de couplage pourront être adaptées en conséquence.

Modélisation de la partie *robot* Le traitement du pilotage en position dans l'espace opérationnel est réalisé selon la démarche présentée dans le chapitre 3.4 et illustrée sur la figure 4.11. Les inconnues du problème d'optimisation sous contraintes sont les six positions articulaires pilotées, les données d'entrées sont la vitesse opérationnelle de l'extrémité du pouce et celle de l'un des trois doigts longs dont les mouvements sont couplés. Le choix du doigt qui impose le mouvement d'entrée est choisi en fonction des zones de contacts retenues par l'algorithme de saisie. Pour ce mécanisme redondant, les degrés de liberté pilotés dans l'espace opérationnel sont sélectionnés par la tâche à réaliser.

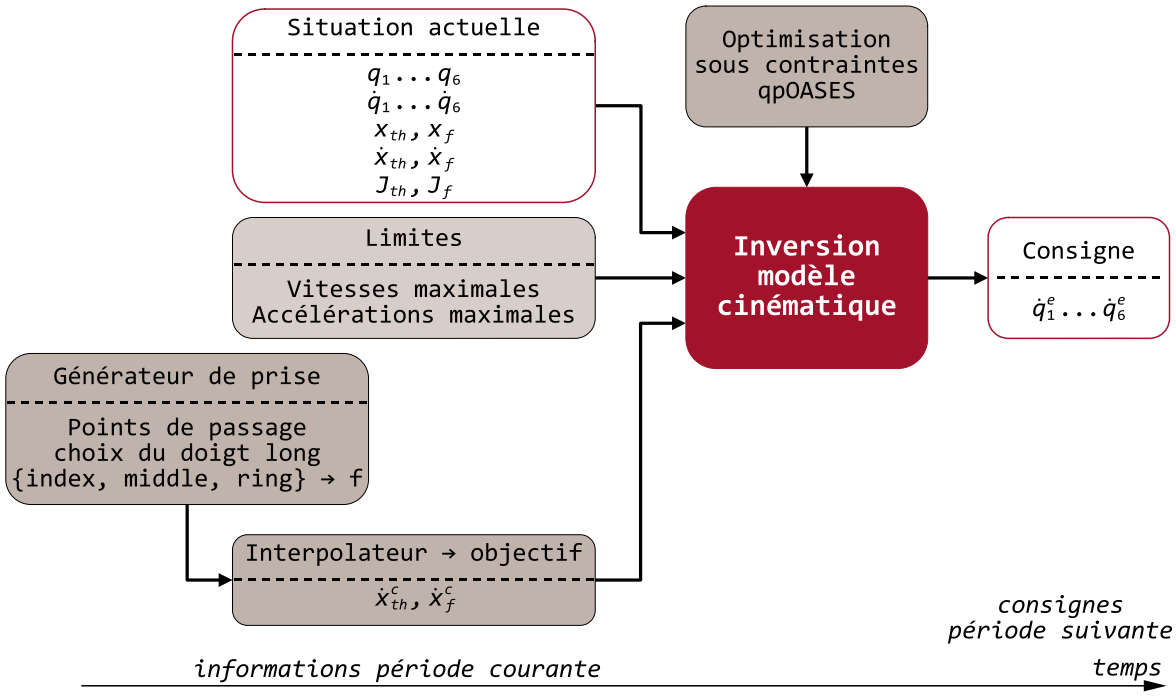


FIGURE 4.11 – Pilotage en position dans l'espace opérationnel.

Pour finir, la figure 4.12 présente les fonctions spécifiques de ce système qui permettent de construire l'état du robot à partir des positions moteurs. Les relations de description du mécanisme permettent de calculer les positions articulaires. Les positions des bouts de doigts x_i et les matrices de rotation absolues des dernières phalanges R_i sont obtenues à partir du modèle géométrique direct, les vitesses des extrémités $\dot{x}_i$ et les vitesses angulaires des segments ω_i, à partir des modèles cinématiques.

La figure 4.13 décrit les relations inverses qui permettent d'obtenir les positions moteurs à partir de consignes de vitesses articulaires ou des vitesses des effecteurs.

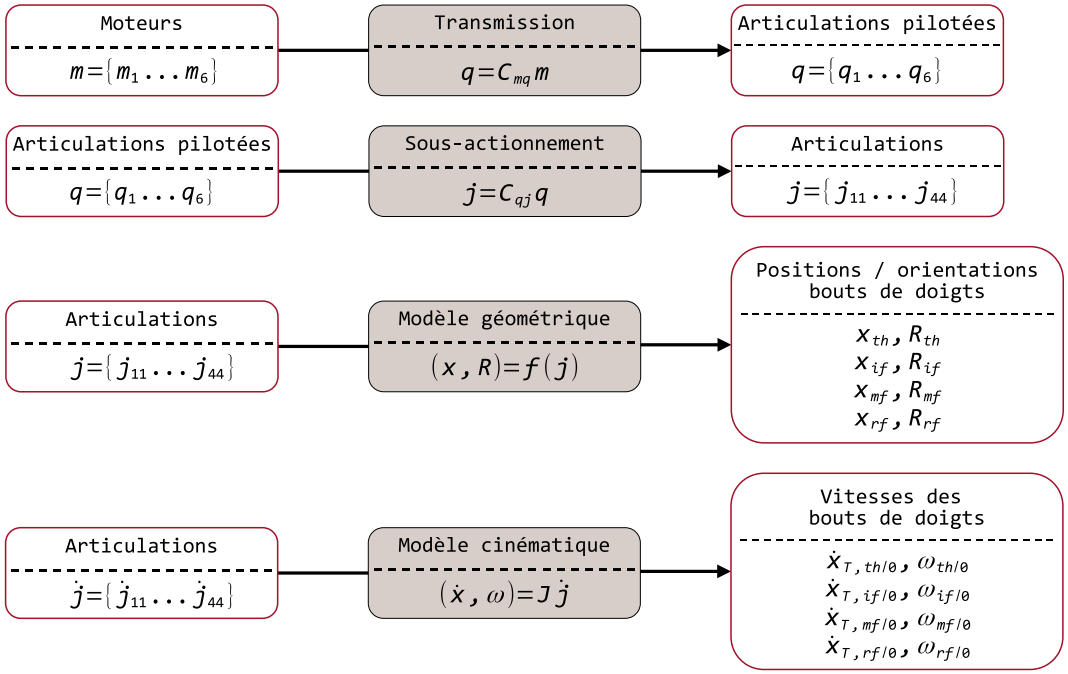


FIGURE 4.12 – Seahand : relations directes, construction de l'état du robot.

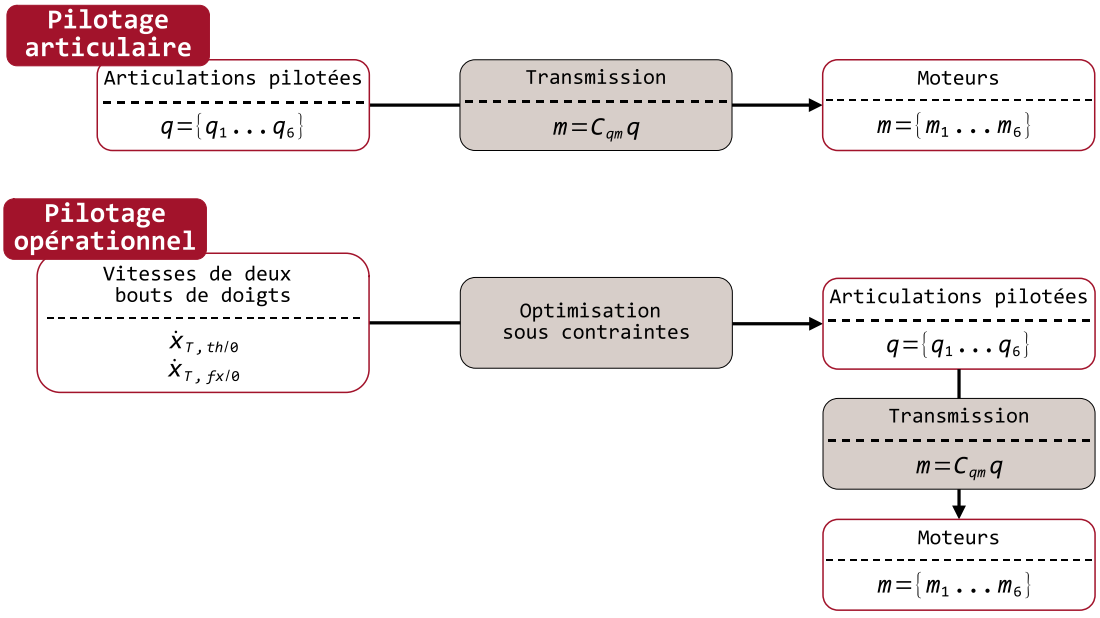


FIGURE 4.13 – Seahand : calcul des consignes instantanées des positions moteurs.

4.4 Développement du contrôle commande et implémentation

La figure 4.14 décrit les différents constituants de la plateforme de développement mise en place pour la réalisation du logiciel de pilotage de la main Seahand. Le logiciel de contrôle est déployé sur deux cibles matérielles : une carte électronique embarquée proche du préhenseur (qui sera dans le sous-marin, dans la version de production) et un automate. La carte électronique va réaliser les fonctions de bas niveau : le conditionnement des signaux des capteurs et l'alimentation des moteurs et exécuter une partie des fonctions de contrôle. L'automate va exécuter les fonctions de plus haut niveau et communiquer avec la carte électronique et le jumeau numérique. Celui-ci va fournir l'interface de pilotage à l'opérateur et calculer les trajectoires permettant la saisie de l'objet à attraper. La saisie d'objets de formes variées à partir d'un préhenseur sous-actionné à plusieurs doigts est un problème complexe. Elle fait l'objet des travaux de thèse de Bastien Arnaud dans l'équipe RoBioSS. Les algorithmes sont développés avec le logiciel Matlab puis exportés sous forme d'une bibliothèque dont les fonctions sont exécutées par le jumeau numérique.

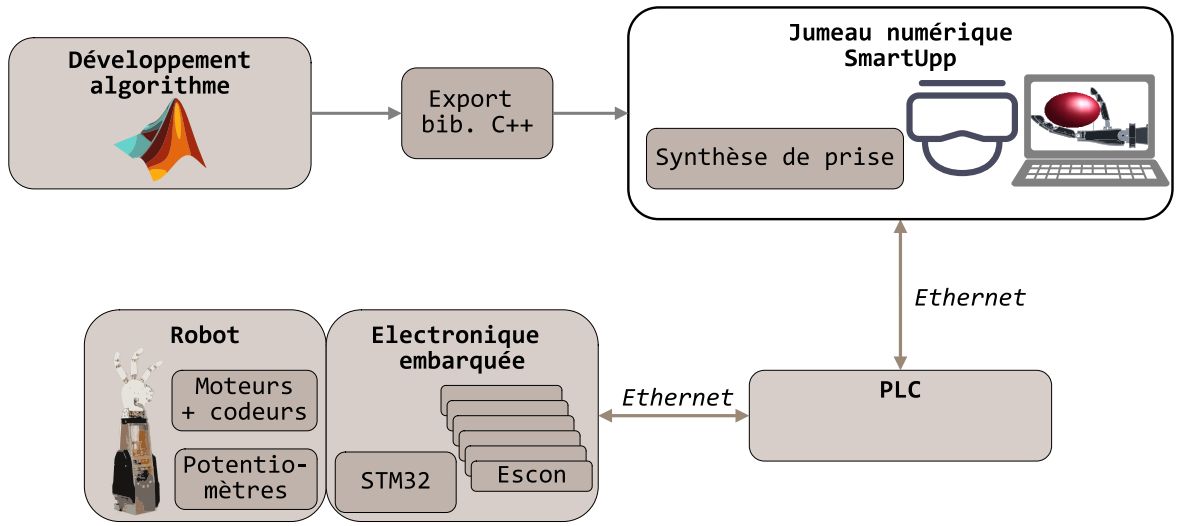


FIGURE 4.14 – Présentation des principaux éléments du pilotage de la main Seahand.

4.4.1 Architectures de contrôle-commande

Le développement du logiciel de pilotage du préhenseur a été réalisé sur deux dispositifs complémentaires. Le premier est un banc de test destiné à évaluer les performances de manipulation d'un doigt (cf. figure 4.15). L'architecture mécanique et la transmission sont identiques à ceux du système réel, il est entraîné par trois motoréducteurs à courant continu. Cependant, la partie commande est différente de celle du prototype de main pour lequel les contraintes d'encombrement sont très exigeantes. Pour le banc de test, les modules de puissance sont des composants industriels standards.

Le prototype de préhenseur robotique est équipé d'une carte électronique embarquée qui

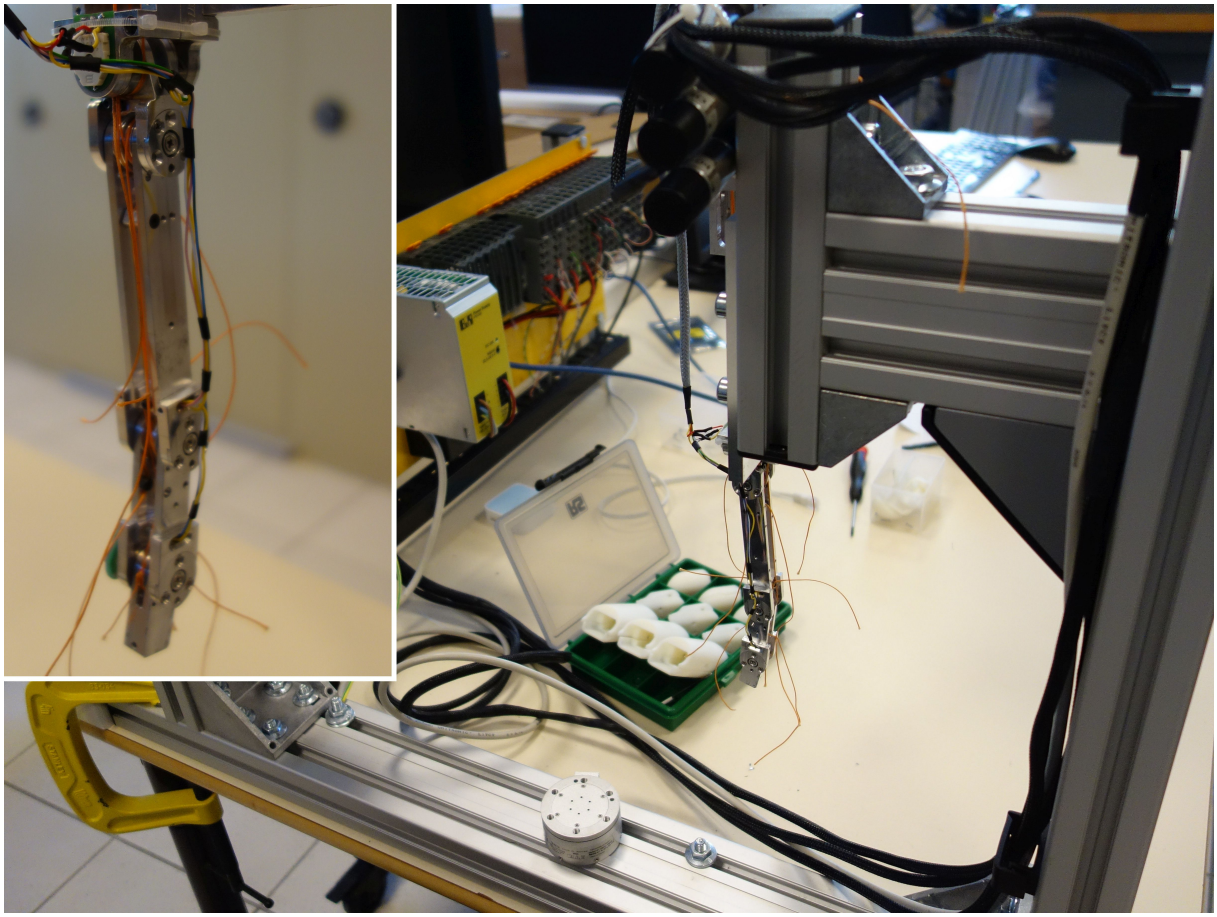


FIGURE 4.15 – Banc de test d'un doigt robotique (vue de détail : doigt de la main Seahand).

réalise une partie du contrôle-commande et la mise en forme de la puissance électrique des six moteurs qui mettent en mouvement les quinze articulations mobiles du mécanisme. La figure 4.16 présente les architectures matérielles des deux dispositifs, banc de test d'un doigt, à gauche et prototype de main, à droite. La figure 4.17 présente la décomposition logicielle pour les deux solutions. L'objectif de cette approche est de mutualiser au maximum les développements et de valider l'aptitude à la mise à l'échelle de la stratégie de développement présentée dans ce manuscrit. Les spécificités matérielles sont prises en compte par des tâches de régulation différentes.

L'adaptation des gestionnaires moteurs aux deux schémas de régulation est prise en charge par la bibliothèque `rtrmac`, le code de ces programmes est identique. Le gestionnaire de robot est également le même dans les deux cas, la spécificité de chaque robot (un seul robot sériel dans le premier cas, un groupe de doigts dans le second cas) est définie dans les classes associées aux objets programmables logiciels qui décrivent le comportement des robots *doigt* et *main*. La taille des espaces des actionneurs, des articulations, et de l'espace opérationnels sont différents. Les fonctions de couplage, le modèle géométrique direct et les modèles cinématiques sont définies pour les deux robots sous la forme de classes dérivées de la classe `RbMC_Robot` présentée sur la

figure 3.18. Ces fonctions sont définies à partir du modèle présenté au chapitre précédent.

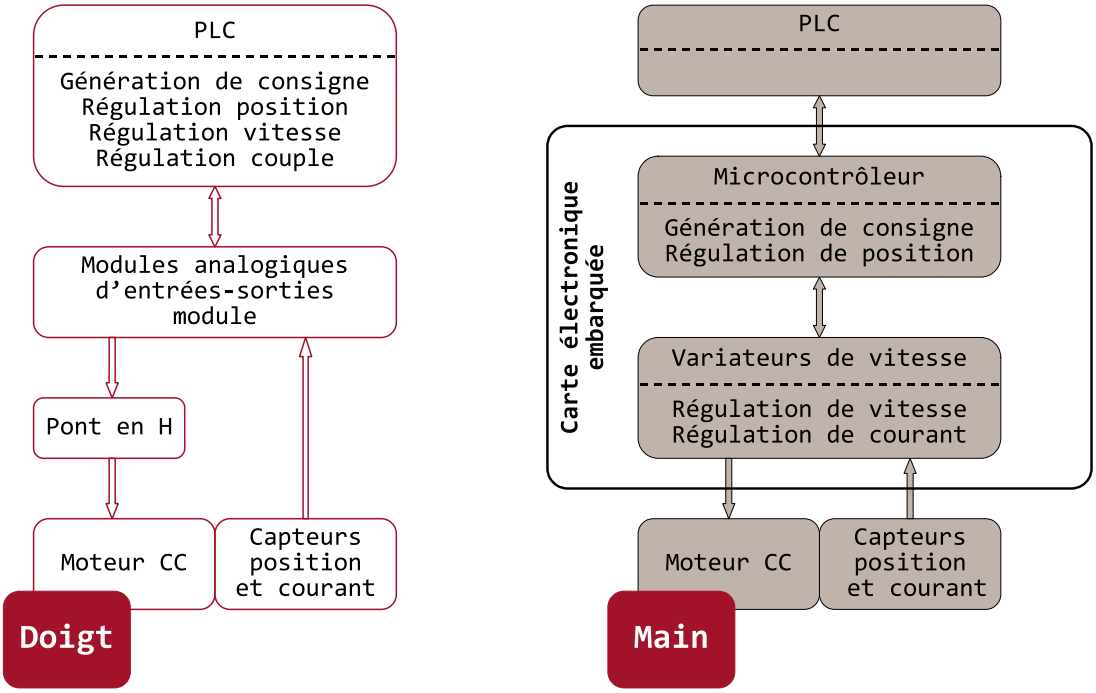


FIGURE 4.16 – Comparaison des architectures matérielles entre le banc de test d'un doigt et la partie commande de la main Seahand et répartition des fonctions de pilotage moteur sur les différents composants.

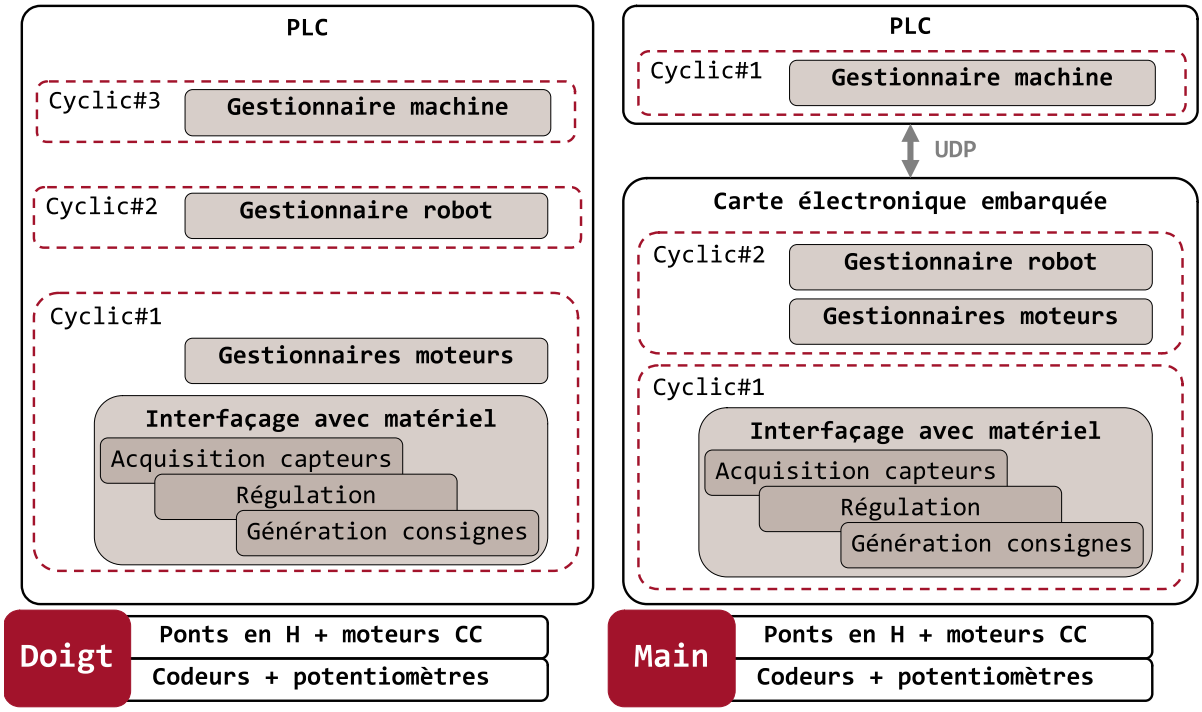


FIGURE 4.17 – Comparaison de l'implémentation du logiciel de commande entre le banc de test d'un doigt et celle prévue sur la partie commande de la main Seahand.

4.4.2 Développement du logiciel de pilotage

Le développement du logiciel de contrôle-commande de la main a été réalisé dans l'environnement de programmation Automation Studio avec une configuration de simulation basée sur un automate virtuel et un jumeau numérique.

Pour évaluer les performances de l'environnement `rtrmac` pour le contrôle des robots, l'architecture de contrôle dans l'espace opérationnel présentée figure 3.19 a été implémentée sur une configuration de simulation, exécutée sur un automate B&R X20CP1586. Les résultats présentés ici portent sur la réalisation d'une séquence de déplacements interpolés dans l'espace opérationnel. [2]

Les points de passage sont définis sur la figure 4.18. Depuis la position de départ A, le TCP du pouce se déplace en ligne droite vers cinq points d'arrêt ($B_1 - B_2 - B_3 - B_4 - B_1$) puis le pouce revient à la position de départ avec un déplacement articulaire. Dans la première phase, la trajectoire attendue est réalisée par la librairie *Reflexxes* qui détermine une consigne instantanée de vitesse du TCP. L'inversion du modèle cinématique est réalisée par la solution à chaque pas de temps d'un problème d'optimisation par la librairie *qpOASES*. Les vitesses articulaires sont converties en consignes de déplacement moteur par le modèle de transmission. Les modèles d'axes fonctionnent en simulation et retournent les positions courantes des moteurs. Les positions articulaires et la position du TCP sont calculées par le modèle de transmission

2. Ce mouvement est présenté en vidéo sur src/koda.cnrs.fr

et le modèle géométrique directs. L'ensemble des fonctions de la chaîne de pilotage sont donc utilisées.

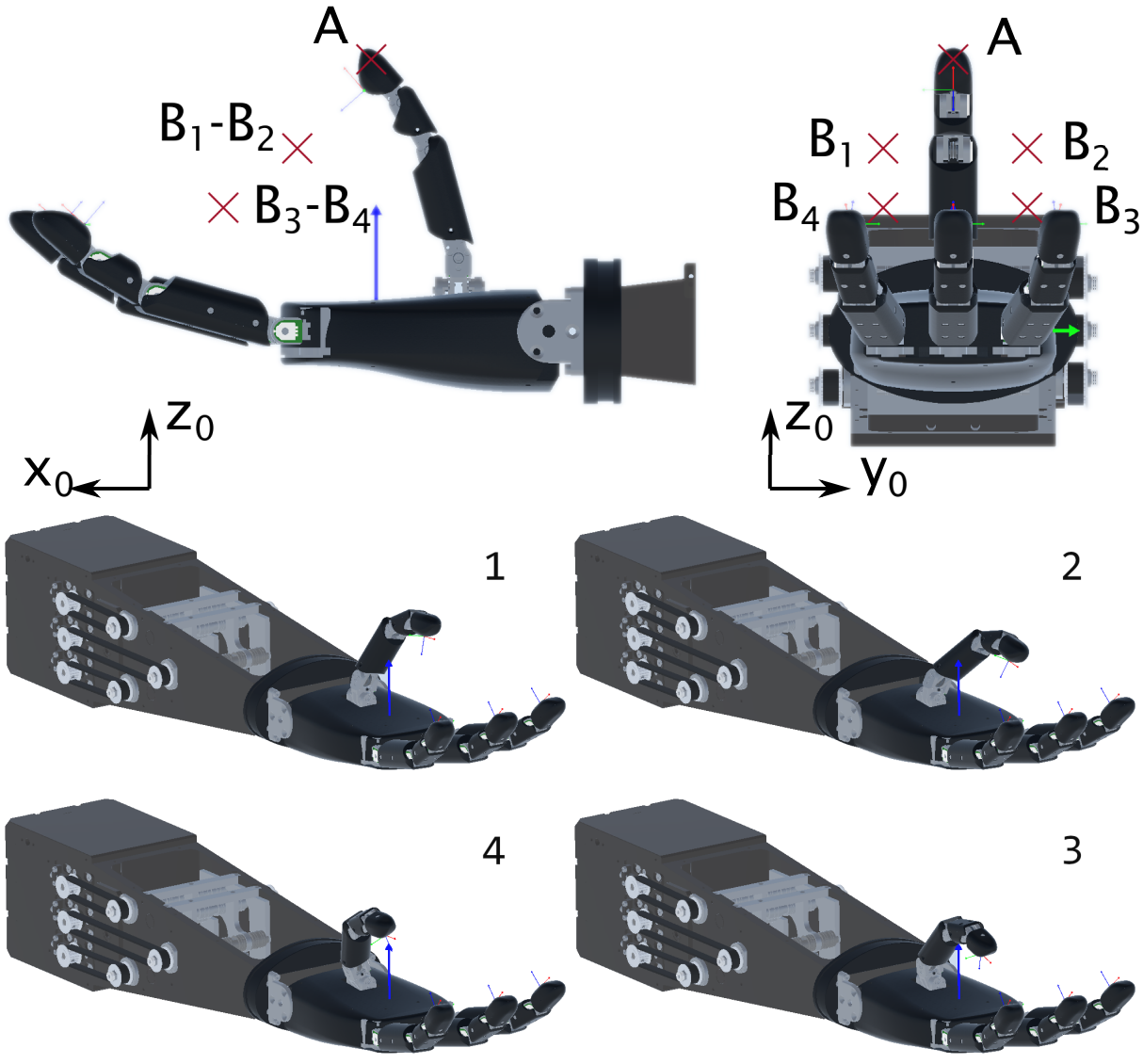


FIGURE 4.18 – Points de passage de la trajectoire à effectuer et capture de la séquence simulée avec le jumeau numérique.

La table 4.4 résume la durée d'exécution des tâches et la charge du processeur obtenues par l'utilisation du *profiler* fourni par le constructeur du matériel. Ce logiciel embarqué sur le PLC enregistre la répartition d'attribution du temps de travail du processeur aux programmes utilisateur et aux tâches systèmes. Tous les programmes de simulation de la régulation, de gestion des axes et du robot sont exécutés dans des tâches associées à la classe de tâches principale. Sa période d'exécution est de 1,6ms, sans débordement autorisé. La charge de travail est peu soutenue puisque le temps libre du processeur est de l'ordre de 50%. La tâche de contrôle du robot qui exécute les modèles directs, l'interpolation dans l'espace opérationnel et l'inversion du

modèle cinématique, a eu, sur le temps d'observation du profiler, une durée moyenne de 290 µs et une durée maximale de 317 µs.

Ces résultats sont obtenus avec un automate X20CP1586 qui est un composant moyen de gamme, destiné à des usages universels de contrôle de machines de production. Ses performances sont modestes au regard des composants généralement utilisés pour le pilotage de robots : il est équipé d'un processeur Intel Atom E680T à 1,6 GHz avec 512 Mo de SDRAM.

TABLE 4.4 – Rapport d'inspection du *profiler* de l'automate pendant l'exécution de la trajectoire test.

Classe de tâches / tâches	% charge CPU	Durée nette moy. [µs]	Durée nette max. [µs]
Idle tasks	48		
Cyclic #1 $[1,6ms]$			
Timer	0.2	3	4
Hardware	2.2	35	41
Gest. axes	1.6	26	29
Gest. robot	18	290	317
Cyclic #2 $[4ms]$			
Gest. machine	0.2	7	8
Com. UDP	0.3	11	17

4.4.3 Implémentation du contrôle-commande sur le prototype

La mise au point du prototype de préhenseur Seahand est en cours de réalisation. La décomposition fonctionnelle du contrôle-commande prévue initialement a dû être adaptée aux performances de la carte électronique embarquée. Celle-ci est basée sur un processeur STM32G474VET sur lequel le système d'exploitation FreeRTOS est installé au-dessus du runtime STM32Cube HAL du processeur STM32. Le runtime fournit les pilotes matériels, les bibliothèques et les services logiciels nécessaires pour interagir avec le microcontrôleur et ses entrées-sorties. FreeRTOS est un système d'exploitation léger qui prend en charge la gestion des tâches utilisateurs avec un ordonnancement périodique qui permet de se placer dans une organisation logicielle similaire à celle des automates programmables industriels. Plusieurs classes de tâches sont définies et sont exécutées sur interruption périodiques de timers système. La bibliothèque *rtrmac* est importée et les programmes utilisateurs développés dans l'environnement Automation Studio en C++ sont directement copiés dans STM32CubeIDE, l'environnement de développement associé au processeur utilisé.

La figure 4.19 présente l'architecture logicielle mise en œuvre sur le démonstrateur. Le comportement cyclique et périodique est réalisé par l'exécution de fonctions sur interruptions. Le groupe d'objets logiciels programmables `Cyclic #1` est associé à un timer de 4 ms, le groupe `Cyclic #2` à un timer de 20ms.

Pour la période d'exécution la plus rapide, la carte de contrôle réalise la mise en forme des données des codeurs, la régulation de position et la génération du signal PWM pour les modules

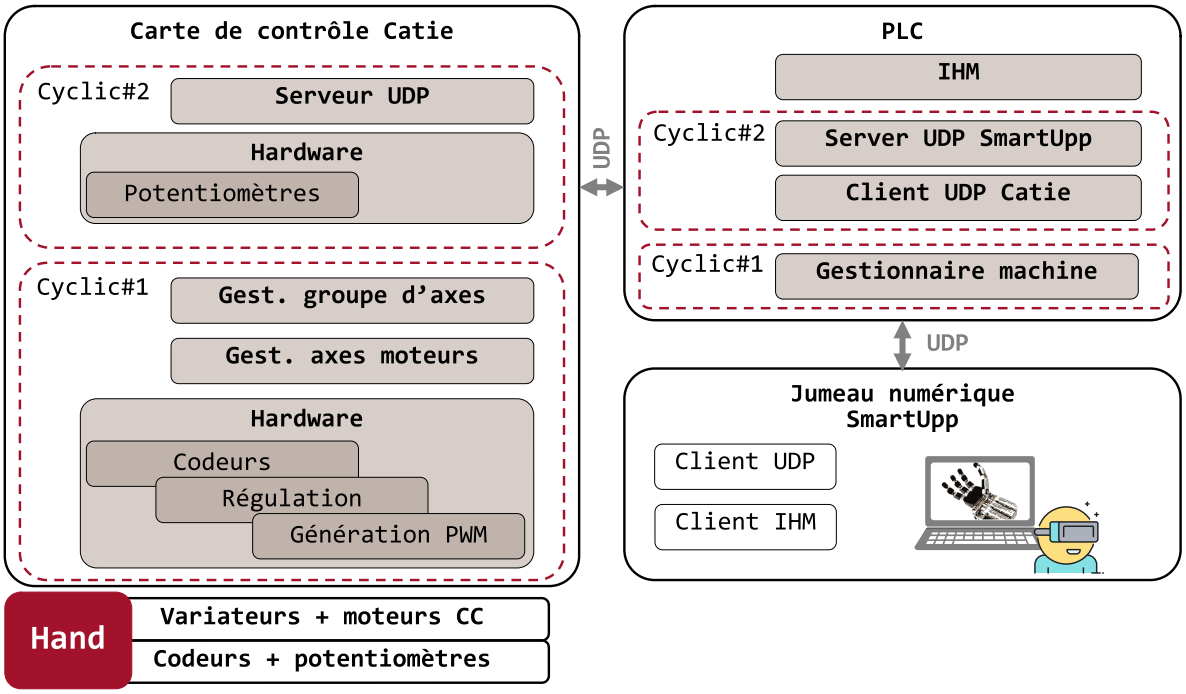


FIGURE 4.19 – Adaptation de l'architecture logicielle aux ressources disponibles.

de commande des six moteurs à courant continu. Elle exécute également les gestionnaires d'axes moteurs du standard PLCopen Motion Control et un programme de coordination qui gère le groupe d'axes moteurs. Dans le groupe `Cyclic #2`, la carte traite les données issues des potentiomètres qui donnent l'estimation des positions de toutes les articulations de la main et met en place la communication UDP avec l'automate de gestion du robot. Automate et carte de contrôle échangent une structure d'information, renseignée par la carte et une structure de commande définie par l'automate. Un mécanisme d'échange de compteurs de vie permet de s'assurer de l'activité de la communication entre les deux composants.

L'automate communique également avec le jumeau numérique de la main qui anime une représentation numérique de la main, exécute l'interface homme-machine du préhenseur robotique et exécute les algorithmes de planification de la saisie. La spécification de la prise en main à distance par le jumeau numérique est définie sur le GRAFCET partiel de la figure 4.20. La variable `EnRemoteCtrl` est la variable permettant d'activer la prise en main à distance via l'IHM exécutée par l'automate. La variable `ComOK` évalue l'activité et la qualité de la communication UDP. L'étape `35` correspond à la situation d'arrêt due à un dépassement de l'écart admissible entre la situation réelle du préhenseur et la commande.

La stratégie de commande mise en place pour cette première version du prototype est la suivante[3] :

— l'objet à saisir est placé par l'utilisateur du jumeau numérique dans le repère de la paume de la main,

3. Cette démarche est présentée en vidéo sur src/koda.cnrs.fr

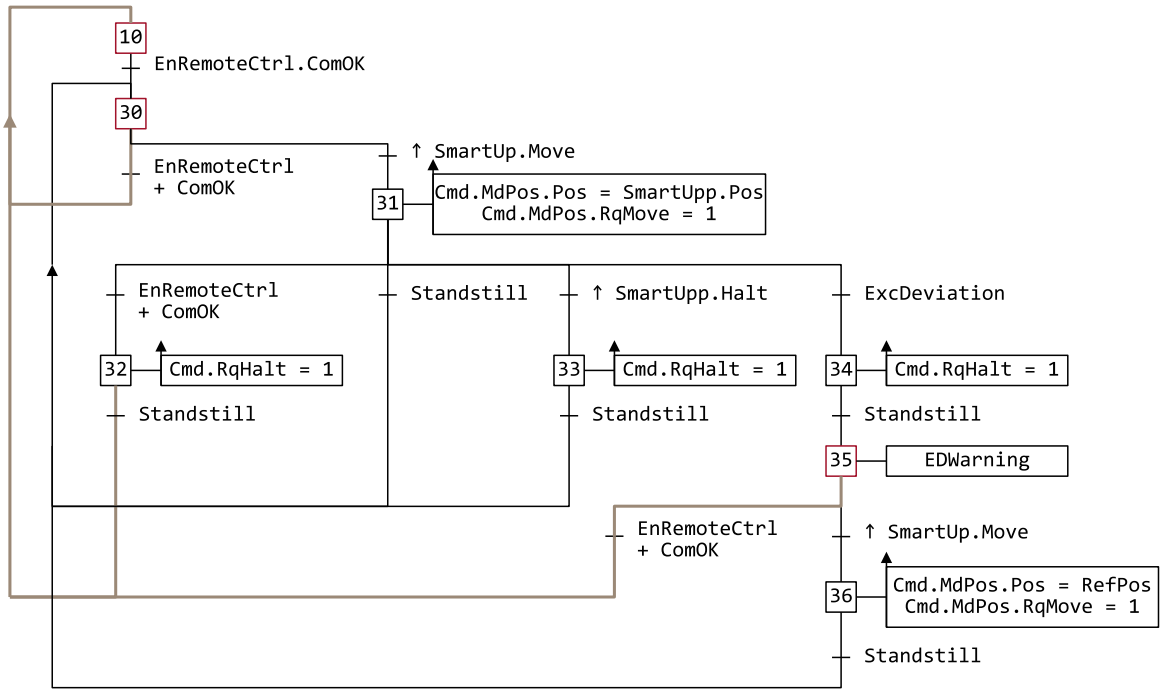


FIGURE 4.20 – Extrait de la spécification du gestionnaire de robot : prise en main distante par le jumeau numérique.

— un algorithme de saisie est exécuté par le jumeau numérique pour déterminer les coordonnées articulaires permettant une saisie robuste,
— le jumeau numérique transmet au gestionnaire de machine exécuté par l'automate les coordonnées articulaires à atteindre et demande l'exécution du mouvement,
— la carte de contrôle exécute le déplacement coordonné dans l'espace articulaire et évalue l'écart entre les coordonnées articulaires mesurées et la position théorique imposée par les positions moteurs courantes. Si l'écart entre les positions théoriques et réelles dépasse une valeur limite, la fermeture du préhenseur s'arrête et l'utilisateur doit acquitter la demande d'ouverture du préhenseur.

4.4.3.1 Etat d'avancement

Le prototype est fonctionnel et a été mis en service à partir du logiciel développé en simulation en parallèle de l'installation et du paramétrage du système d'exploitation de la carte électronique. Le principe d'une plateforme de développement constituée de la bibliothèque *rtrmac*, du jumeau numérique *SmartUpp* et de *Matlab* a été mise en œuvre : le développement des algorithmes de saisie dans le cas d'un préhenseurs sous-actionné à plusieurs doigts est réalisé sous Matlab. Les fonctions qui convertissent la position et la géométrie de l'objet sont exportées en C++ et exécutée par le jumeau numérique. Celui-ci permet de faire changer les propriétés des objets saisis et transmet les positions articulaires de la séquence permettant la prise de l'objet à l'automate qui exécute le programme de gestion machine et délègue à la carte électronique la gestion des

axes moteurs.

Les tâches en cours de réalisation portent sur :

— la correction de défauts de conception mécanique pour augmenter la robustesse des transmissions par câbles,
— la modélisation du comportement visco-élastique des câbles et son intégration dans la boucle de commande,
— la simplification de la couche de gestion du mécanisme (l'étage de liaison moteurs - articulations pilotées) pour pouvoir l'implémenter dans la carte électronique embarquée,
— le choix d'interfaces et la recherche de stratégies de téléopération de la main robotique par un opérateur en situation de conduite en réalité virtuelle,
— et, d'une façon plus générale, l'évaluation des performances globales du préhenseur.

Conclusion

Le travail d'unification de la conception des logiciels de nos dispositifs mécatroniques s'est construit sur la synthèse de nombreux projets antérieurs. La version en cours de notre bibliothèque de contrôle-commande est actuellement utilisée sur six projets en parallèle et les retours d'expérience des différents cas d'usage sont en cours d'analyse.

La démarche retenue pour décrire les mécanismes et les architectures robotiques est adaptée aux mécanismes rencontrés. La diminution du temps de développement des nouveaux logiciels obtenue est compatible avec l'évolution du nombre de systèmes à développer.

L'un des objectifs majeurs de cette étude est le partage des responsabilités de développement. Cet aspect n'est pas encore évalué. La décomposition fonctionnelle retenue permet de confier à un automaticien les tâches de paramétrage des composants matériels. La réalisation des fonctions spécifiques pour un nouveau robot ou l'implémentation de nouveaux algorithmes peut être confiée à un chercheur ou un ingénieur sans prérequis de compétences en contrôle-commande industriel.

Les perspectives de mise en œuvre de ce travail sont nombreuses. Les choses sérieuses peuvent vraiment commencer. Les premiers développements que nous allons réaliser portent sur plusieurs aspects.

Modélisation des mécanismes Certains systèmes mécatroniques que nous avons conçus présentent des transmissions de mouvement dont le comportement doit être représenté finement pour pouvoir exploiter au mieux les dispositifs. C'est le cas des systèmes à transmission par câbles pour lesquels la modélisation des lois de comportement des câbles est nécessaire pour réaliser certaines fonctions. Pour les préhenseurs, cette étape est nécessaire pour évaluer les efforts de saisie et garantir le niveau de serrage des prises. L'extension des modes de pilotage au pilotage en couple articulaire est également un objectif scientifique pour lequel le travail de modélisation a été réalisé.

Intégration de nouvelles lois de contrôle Pour la partie contrôle haut-niveau, de nouveaux problèmes de calcul instantané de lois de consignes articulaires peuvent être développés. En particulier, des approches de limitation de l'énergie cinétique d'un robot en présence d'un opérateur ont été développées dans le cadre du projet européen COVR. Le travail réalisé dans cette thèse permet d'intégrer ces algorithmes dans le calculateur industriel pour minimiser les temps de réaction du système.

Modèles dynamiques La bibliothèque de contrôle multiaxe peut également être étendue pour intégrer des modèles dynamiques permettant d'évaluer des efforts d'interaction ou d'intégrer l'estimation des couples moteurs dans la régulation des axes. Des essais préliminaires ont montré l'intérêt d'utiliser des modèles spécifiques aux robots générés par des modèles paramétriques résolus par une approche formelle [Riccoboni 2022].

Nouvelles architectures mécaniques Enfin, de nombreux systèmes sont en cours de développement et vont pouvoir élargir la diversité des situations d'usage de notre solution de contrôle

multiaxe. Les prochaines applications vont mettre en œuvre des bras robotiques industriels dans des cellules à six, sept et treize axes coordonnés. Le portage de la bibliothèque *rtrmac* sur une nouvelle gamme de composants industriels est également en cours d'évaluation.

Annexes

Annexe A

Gestionnaire de mécanisme : phases de préparation

Une représentation simplifiée de la machine d'états des classes de gestion de mécanisme ou de robot est présentée sur la figure A.1. L'état `ACCESS_OK` correspond à la situation initiale, si tous les gestionnaires d'axes virtuels et réels peuvent être démarrés, l'état `DORMANT` s'active. Le mécanisme peut être ensuite démarré et référencé (ou référencé puis démarré). L'état où il est disponible pour démarrer un mouvement est l'état `STANDSTILL`. Les commandes `EnPowerOn`, `RqHome`, `EnStop` sont des champs de la structure de commande de type `RbRobCmd_typ`. Ce sont des commandes à deux niveaux (de type *Enable* avec le préfixe `En`, ou à l'activation sur front montant de type `Request`). Les informations `PowerOn` et `IsHomed` sont des champs de la structure de type `RbRobInfo_typ`, elles indiquent tous les axes ont démarré, et que toutes les prises d'origines sont faites, respectivement. L'activation de la commande `EnStop` provoque un arrêt d'urgence logiciel qui réalise un freinage rapide sur les axes articulaires. Lorsque cette commande est désactivée, la machine d'états évolue vers `STANDSTILL` si le couplage est toujours actif ou tous les axes sont arrêtés, le couplage est rompu et la machine d'états se place en `DORMANT`.

La figure A.2 présente plus précisément la réalisation des étapes de prises d'origine et de couplage. Suivant les mécanismes, la situation qui permet d'associer l'information des codeurs avec une configuration réelle peut être définie dans l'espace des moteurs ou dans l'espace des articulations. En général, la phase de calibration est réalisée sur des références physiques (repères, butées, information d'un capteur de position) liées aux articulations. Par contre, si les moteurs sont équipés de capteurs absolus ou si les valeurs des données des capteurs en position d'arrêt sont mémorisées, les prises d'origines suivantes sont réalisées à partir des informations des capteurs de position des moteurs. Ces deux possibilités se traduisent par des parcours différents de la machine d'états selon l'état de la variable `HomingFromAcsPos` qui est vraie lorsque la prise d'origine dans l'espace articulaire.

Lorsque les axes moteurs et les axes articulaires virtuels ont démarré et que la prise d'origine est faite, les axes moteurs passent en mode de suivi de consigne de position et le couplage est actif. A partir de ce moment, les consignes de position des moteurs sont calculées à partir des positions instantanées des axes articulaires par la méthode `AcsToMot` qui doit être définie pour chaque classe de mécanisme qui dérive de la classe abstraite `RbMC_Mechanism`.

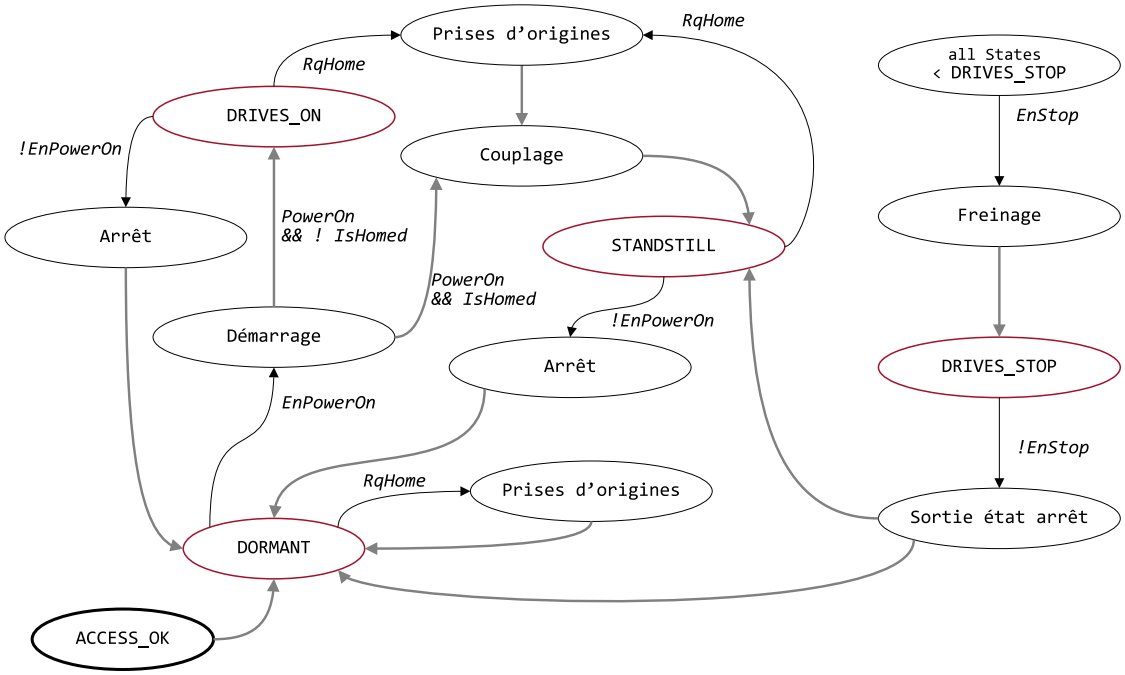


FIGURE A.1 – Représentation simplifiée des phases de préparation d'un mécanisme ou d'un robot.

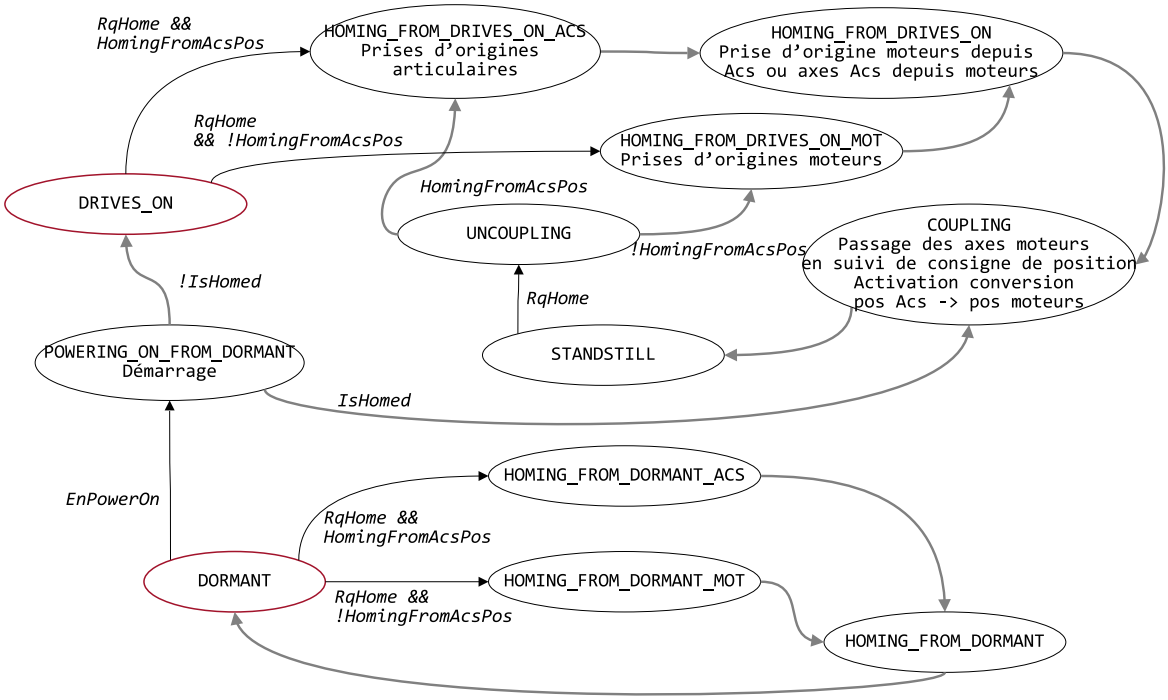


FIGURE A.2 – Vue détaillée de la machine d'états des phases de prise d'origine et de couplage des axes moteurs et articulaires.

Annexe B

Main Seahand : modélisation

B.1 Mécanisme

La figure B.1 présente les relations définissant la transmission entre les six moteurs et les six articulations pilotées.

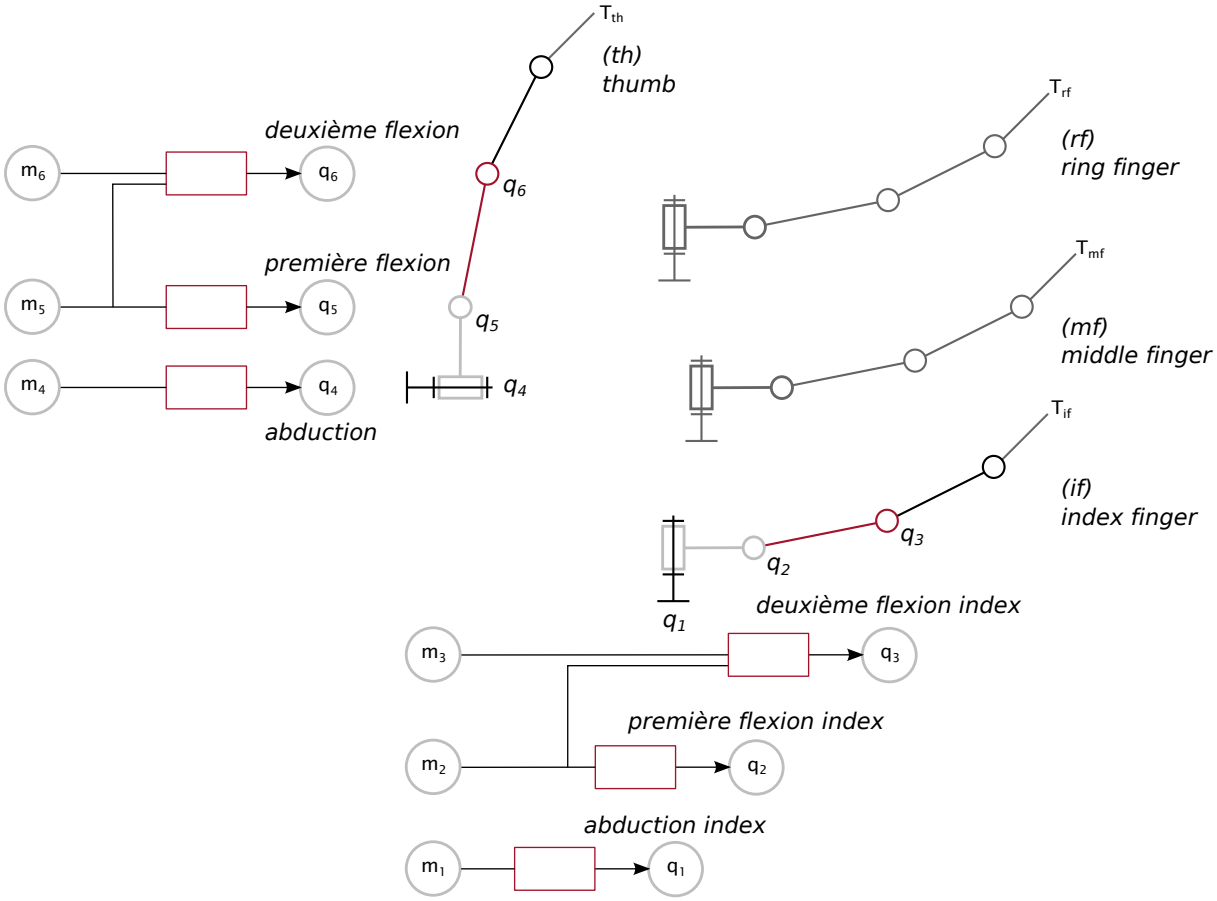


Figure B.1 – Main SeaHand, description du mécanisme. Conversion des positions moteurs en positions articulaires pilotées.

La matrice C_{mq} relie, pour un doigt, les variations de positions moteurs (m) aux variations de positions des articulations pilotées (q).

$$C_{ma} = \begin{pmatrix} \frac{D_{m1}}{D_{r1}} & 0 & 0 \\ 0 & \frac{D_{m2}}{D_{r2}} & 0 \\ 0 & -\frac{D_{m2}}{D_{r2}} \cdot \frac{D_{i23}}{D_{r3}} & \frac{D_{m3}}{D_{r3}} \end{pmatrix}$$

La figure B.2 illustre les relations les positions des articulations pilotées (*ACS*, notées q) et celles de l'ensemble des articulations (*Joints*, notées j).

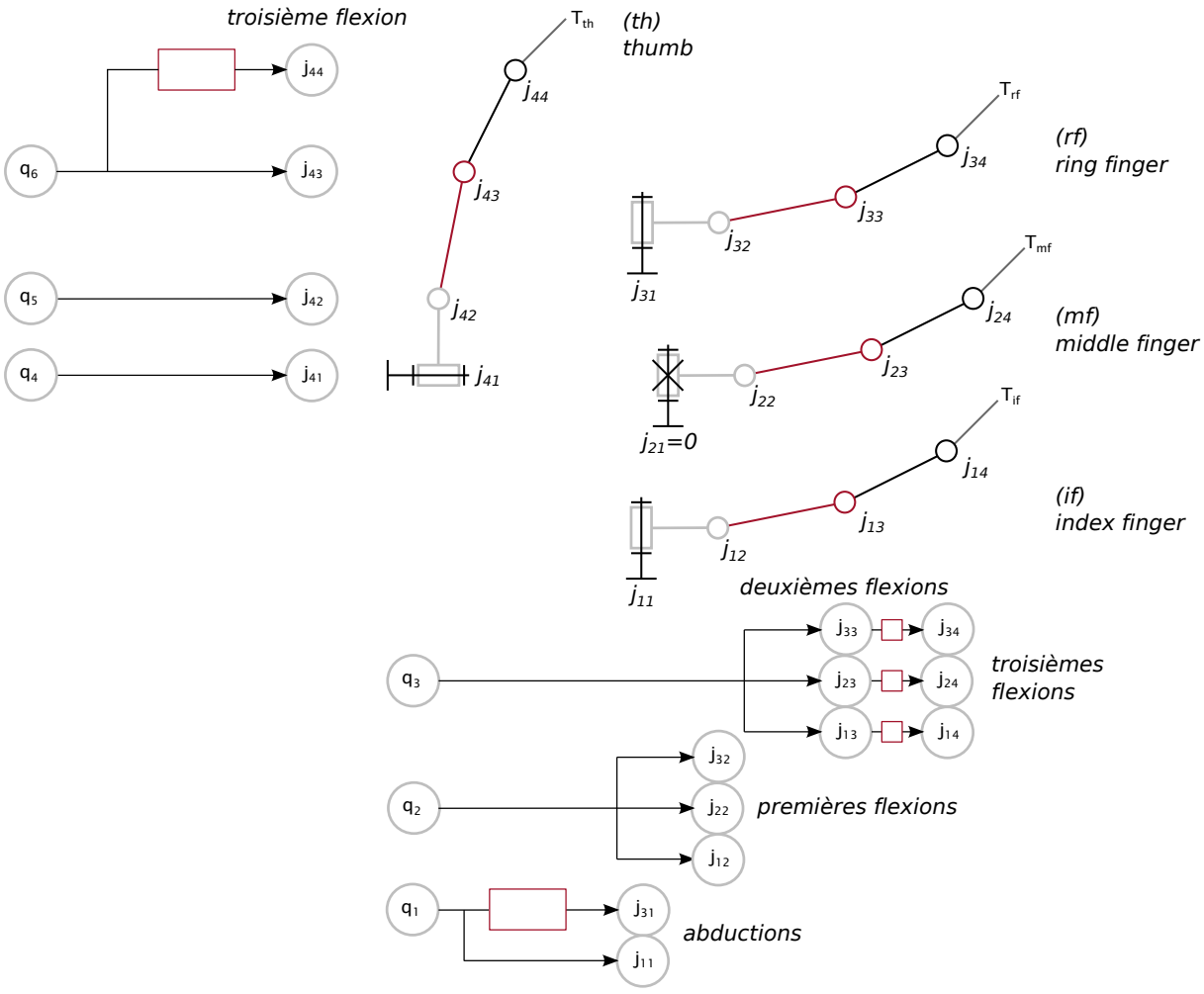


FIGURE B.2 – Main SeaHand, description du mécanisme. Conversion des positions articulaires pilotées en positions articulaires.

Les relations suivantes mettent en place les relations de couplage entre les articulations pilotées et l'ensemble des articulations pour le premier doigt long. La dernière flexion n'est pas directement actionnée, elle est liée par une relation de couplage aux deux précédentes flexions.

$$C_{qj} = \begin{pmatrix} 1 & 0 & 0 & \\ 0 & 1 & 0 & (0)_{4\times 3} \\ 0 & 0 & 1 & \\ 0 & -\frac{D_{m2}}{D_{r2}} \cdot \frac{D_{i23}}{D_{r3}} \cdot \frac{D_{i34}}{D_{r4}} & \frac{D_{m3}}{D_{r3}} \cdot \frac{D_{i34}}{D_{r4}} & \end{pmatrix}$$

TABLE B.1 – Éléments caractéristiques de la transmission de mouvement pour la main SeaHand.

D_{m1}	D_{m2}	D_{m3}	D_{r1}	D_{r2}	D_{r3}	D_{i23}	D_{r4}	D_{i34}
16 mm	16 mm	16 mm	16 mm	16 mm	14 mm	14 mm	14 mm	10.5 mm

La géométrie de la main SeaHand est définie par les tailles de poulies présentées dans la table B.1. Dans cette configuration, pour le pouce et l'index (repérés th et if), les matrices de transmission directe et inverse se simplifient de la façon suivante :

$$C_{mq,th} = C_{mq,if} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & t_{33} \end{pmatrix}$$

avec $t_{33} = 8/7$.

$$C_{qm,th} = C_{qm,if} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1/t_{33} & 1/t_{33} \end{pmatrix}$$

La matrice de couplage qui permet d'obtenir la position de toutes les articulations est la suivante :

$$C_{qj} = \begin{pmatrix} 1 & 0 & 0 & & & \\ 0 & 1 & 0 & & & \\ 0 & 0 & 1 & & & \\ 0 & 0 & k & & & \\ 0 & 0 & 0 & & & \\ 0 & 1 & 0 & & & \\ 0 & 0 & 1 & & (0)_{12\times 3} & \\ 0 & 0 & k & & & \\ -1 & 0 & 0 & & & \\ 0 & 1 & 0 & & & \\ 0 & 0 & 1 & & & \\ 0 & 0 & k & & & \\ & & & 1 & 0 & 0 \\ & (0)_{4\times 3} & & 0 & 1 & 0 \\ & & & 0 & 0 & 1 \\ & & & 0 & 0 & k \end{pmatrix}$$

avec $k = 0,75$, qui relie directement les troisièmes flexions aux deuxièmes.

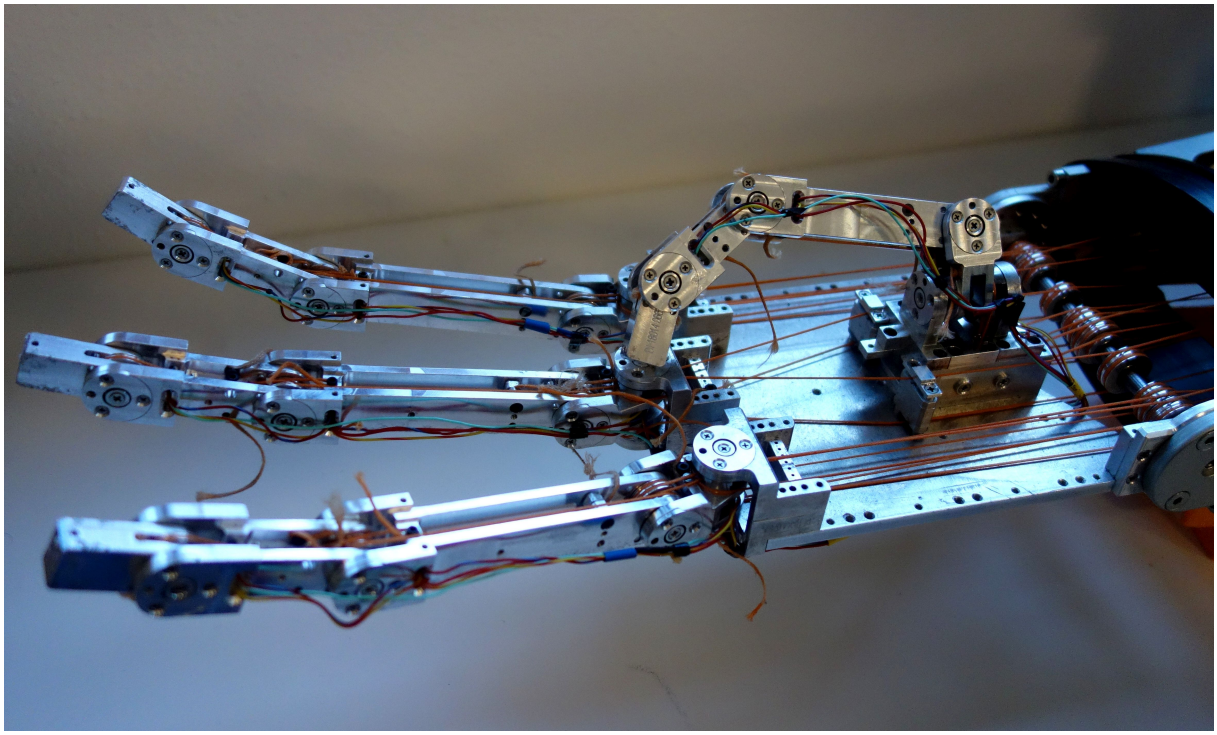


FIGURE B.3 – Vue partielle des transmissions par câbles.

B.2 Robot

Le paramétrage associé à un doigt est illustré sur la figure B.4. Les matrices de passage homogènes entre les repères associés aux différents solides sont les suivantes (on utilise les conventions $C1 = \cos(q_1)$, $S_1 = \sin(q_1)$) :

$$T^{01} = \begin{pmatrix} C1 & -S1 & 0 & 0 \\ S1 & C1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T^{12} = \begin{pmatrix} C2 & -S2 & 0 & L_1 \\ 0 & 0 & -1 & 0 \\ S2 & C2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T^{23} = \begin{pmatrix} C3 & -S3 & 0 & L_2 \\ S3 & C3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T^{34} = \begin{pmatrix} C4 & -S4 & 0 & L_3 \\ S4 & C4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

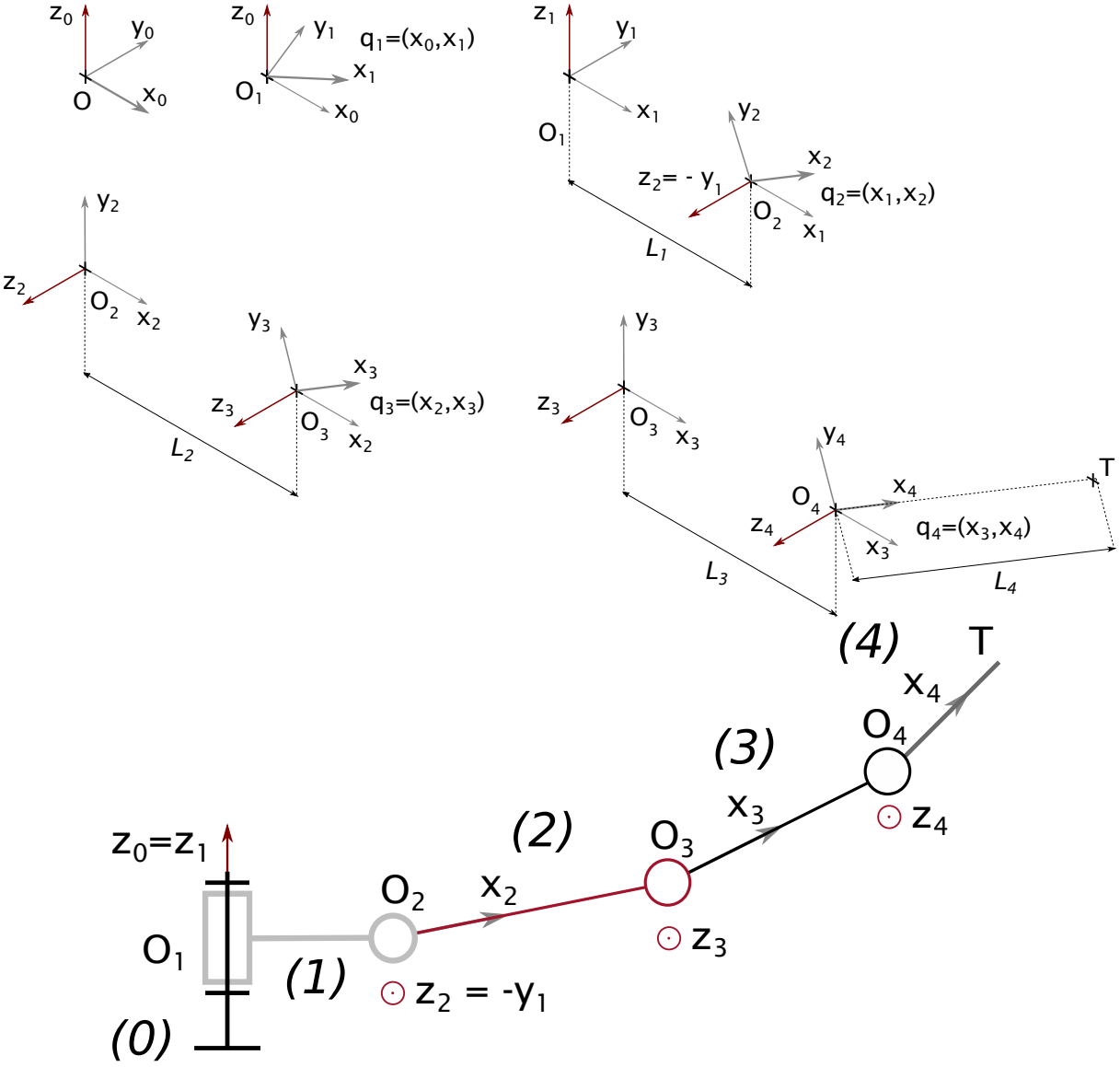


FIGURE B.4 – Main SeaHand. Paramétrage d'un doigt.

$$T^{45} = \begin{pmatrix} 1 & 0 & 0 & L_4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Le placement des doigts dans la paume est présenté sur la figure B.5. Le centre du repère robot O est au milieu de la paume.

Chaque doigt ne disposant que de trois degrés de liberté pilotés (abduction-adduction, première et deuxième flexions), les six degrés de liberté de la phalange terminale ne peuvent pas être contrôlés de façon indépendante. Les directions de l'espace opérationnel contrôlables choisies sont les trois composantes des vecteurs vitesses des TCP de l'index et du pouce. La matrice jacobienne naturelle du préhenseur regroupe deux matrices élémentaires indépendantes, l'une

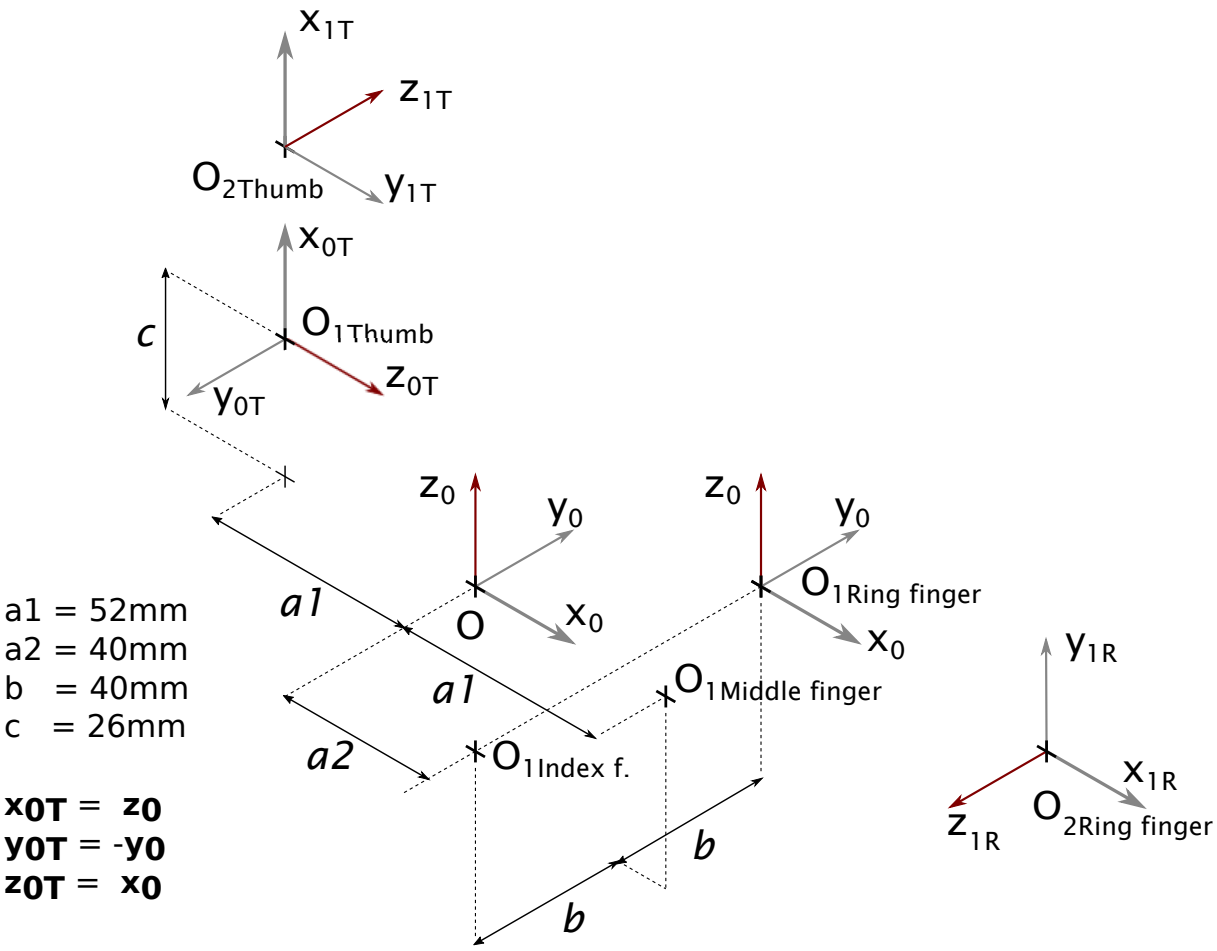


FIGURE B.5 – Main SeaHand. Placement des quatre doigts sur la paume.

pour l'index et la seconde pour le pouce. Ces matrices sont telles que :

$$\dot{x}_{if} = J \begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{pmatrix} \quad \dot{x}_{th} = J \begin{pmatrix} \dot{q}_4 \\ \dot{q}_5 \\ \dot{q}_6 \end{pmatrix}$$

$\dot{x}_{if}$ et $\dot{x}_{th}$ sont les vecteurs vitesses des points terminaux des phalanges distales de l'index et du pouce, repérés T sur la figure B.4.

Les matrices jacobiennes de l'index J_{if} et du pouce J_{th} s'expriment de la façon suivante :

$$J_{if}(:,1) = \begin{pmatrix} 0 \\ 0 \\ 1 \\ -S1.(L_1 + L_2.C2 + L_3.C23 + L_4.C234) \\ C1.(L_1 + L_2.C2 + L_3.C23 + L_4.C234) \\ 0 \end{pmatrix}$$

$$J_{if}(:,2) = \begin{pmatrix} S1 \\ -C1 \\ 0 \\ -C1.(L_2.S2 + L_3.S23 + L_4.S234) \\ -S1.(L_2.S2 + L_3.S23 + L_4.S234) \\ L_2.C2 + L_3.C23 + L_4.C234 \end{pmatrix}$$

$$J_{if}(:,3) = \begin{pmatrix} S1 \\ -C1 \\ 0 \\ -C1.(L_3.S23 + L_4.S234) \\ -S1.(L_3.S23 + L_4.S234) \\ L_3.C23 + L_4.C234 \end{pmatrix}$$

$$J_{th}(:,1) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ -C1.(L_1 + L_2.C2 + L_3.C23 + L_4.C234) \\ -S1.(L_1 + L_2.C2 + L_3.C23 + L_4.C234) \end{pmatrix}$$

$$J_{th}(:,2) = \begin{pmatrix} 0 \\ C1 \\ S1 \\ L_2.C2 + L_3.C23 + L_4.C234 \\ S1.(L_2.S2 + L_3.S23 + L_4.S234) \\ -C1.(L_2.S2 + L_3.S23 + L_4.S234) \end{pmatrix}$$

$$J_{th}(:,3) = \begin{pmatrix} 0 \\ C1 \\ S1 \\ L_3.C23 + L_4.C234 \\ S1.(L_3.S23 + L_4.S234) \\ -C1.(L_3.S23 + L_4.S234) \end{pmatrix}$$

Ces relations permettent de particulariser, pour ce robot :

— les fonctions de couplage directe et inverse qui relient les positions moteurs, les articulations pilotés, puis toutes les articulations,

— les fonctions de transformation géométrique directe qui permettent d'obtenir le mouvement des quatre bouts de doigts,

— les éléments qui vont permettre de renseigner le problème d'inversion des modèles cinématiques.

Bibliographie

[AG 2021] Siemens AG. *Specification of Standard Robot Command Interface to PLCs*, 2021. (Cité en page 105.)

[Al-Jaroodi 2012] Jameela Al-Jaroodi et Nader Mohamed. *Service-Oriented Middleware : A Survey*. Journal of Network and Computer Applications, vol. 35, no. 1, pages 211–220, 2012. (Cité en page 23.)

[Andersen 2015] Thomas Timm Andersen. *Optimizing the Universal Robots ROS Driver*. Rapport technique, Technical University of Denmark, Department of Electrical Engineering, 2015. (Cité en page 17.)

[Ando 2005] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku et Woo-Keun Yoon. *RT-Component Object Model in RT-Middleware—Distributed Component Middleware for RT (Robot Technology)*. In International Symposium on Computational Intelligence in Robotics and Automation, pages 457–462, Espoo, Finland, 2005. IEEE. (Cité en pages 27 et 31.)

[Arias 2010] Soraya Arias, Florine Boudin, Roger Pissard-Gibollet, Daniel Simon et INRIA Rhone-Alpes. *ORCCAD, Robot Controller Model and Its Support Using Eclipse Modeling Tools*. In CAR 2010 - 5th National Conference on Control Architecture of Robots,, Douai, France, 2010. (Cité en page 33.)

[Arkin 1997] Ronald C. Arkin et Tucker Balch. *AuRA : Principles and Practice in Review*. Journal of Experimental & Theoretical Artificial Intelligence, vol. 9, no. 2-3, pages 175–189, 1997. (Cité en page 12.)

[Bakken 2001] D.E. Bakken. *Middleware*. In Encyclopedia of Distributed Computing, J. Urban and P. Dasgupta. Kluwer academic, dodrecht édition, 2001. (Cité en page 20.)

[Bertezene 2022] Sandra Bertezene. *Le jumeau numérique en santé : Apports organisationnels et limites épistémologiques dans un contexte de crise sanitaire*. médecine/sciences, vol. 38, no. 8-9, pages 663–668, 2022. (Cité en page 142.)

[Birrell 1983] Andrew D. Birrell et Bruce Jay Nelson. *Implementing Remote Procedure Calls*. In Proceedings of the Ninth ACM Symposium on Operating Systems Principles - SOSP '83, page 3, Bretton Woods, New Hampshire, United States, 1983. ACM Press. (Cité en page 20.)

[Blank 2005] Doug Blank, Deepak Kumar, Lisa Meeden et Holly Yanco. *The Pyro Toolkit for AI and Robotics*. AI Magazine, 2005. (Cité en page 27.)

[Bolton 2018] Ruth N. Bolton, Janet R. McColl-Kennedy, Lilliemay Cheung, Andrew Gallan, Chiara Orsingher, Lars Witell et Mohamed Zaki. *Customer Experience Challenges : Bringing Together Digital, Physical and Social Realms*. Journal of Service Management, vol. 29, no. 5, pages 776–808, 2018. (Cité en page 144.)

[Briod 2014] Adrien Briod, Przemyslaw Kornatowski, Jean-Christophe Zufferey et Dario Floreano. *A Collision-resilient Flying Robot*. Journal of Field Robotics, vol. 31, no. 4, pages 496–509, 2014. (Cité en page 8.)

[Briquet-Kerestedjian 2019] Nolwenn Briquet-Kerestedjian. *Impact Detection and Classification for Safe Physical Human-Robot Interaction under Uncertainties*. PhD thesis, Université Paris Saclay, 2019. (Cité en page 6.)

[Brooks 1986] R. Brooks. *A Robust Layered Control System for a Mobile Robot*. IEEE Journal on Robotics and Automation, vol. 2, no. 1, pages 14–23, 1986. (Cité en page 12.)

[Brooks 2005] A. Brooks, T. Kaupp, A. Makarenko, S. Williams et A. Oreback. *Towards Component-Based Robotics*. In IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 163–168, Edmonton, Alta., Canada, 2005. (Cité en page 31.)

[Bruemmer 2006] David J Bruemmer. *The Robot Intelligence Kernel*. In The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, Boston, Massachusetts, USA, 2006. (Cité en page 32.)

[Bruyninckx 2001] H. Bruyninckx. *Open Robot Control Software : The OROCOS Project*. In Proceedings ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164), volume 3, pages 2523–2528, Seoul, South Korea, 2001. (Cité en pages 27, 31 et 37.)

[Bruyninckx 2003] H. Bruyninckx, P. Soetens et B. Koninckx. *The Real-Time Motion Control Core of the Orocos Project*. In IEEE International Conference on Robotics and Automation (Cat. No.03CH37422), pages 2766–2771, Taipei, Taiwan, 2003. (Cité en page 37.)

[Bruyninckx 2023] Herman Bruyninckx. Building blocks for complicated and situational aware robotic and cyber-physical systems. 2023. (Cité en page 61.)

[Calinon 2017] Sylvain Calinon et Dongheui Lee. *Learning Control*. In Ambarish Goswami et Prahlad Vadakkepat, editeurs, Humanoid Robotics : A Reference, pages 1–52. Springer Netherlands, Dordrecht, 2017. (Cité en page 132.)

[Calisi] Daniele Calisi, Andrea Censi, Luca Iocchi et Daniele Nardi. *OpenRDK : A Modular Framework for Robotic Software Development*. In IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 1872–1877, Nice. (Cité en pages 27, 32 et 34.)

[Capra 2001] L. Capra, W. Emmerich et C. Mascolo. *Middleware for Mobile Computing : Awareness vs. Transparency*. In Proceedings Eighth Workshop on Hot Topics in Operating Systems, page 164, Elmau, Germany, 2001. IEEE Comput. Soc. (Cité en page 23.)

[Caputo 2013] Antonio C. Caputo, Pacifico M. Pelagagge et Paolo Salini. *AHP-based Methodology for Selecting Safety Devices of Industrial Machinery*. Safety Science, vol. 53, pages 202–218, 2013. (Cité en page 5.)

[Chacko 2013] Joseph Chacko, Neil Richards, Erin Schnabel et Katherine Tsui. *OBJECT REQUEST BROKER*, 2013. (Cité en page 21.)

[Chen 2018] Ximing Chen, Eunsuk Kang, Shinichi Shiraishi, Victor M. Preciado et Zhihao Jiang. *Digital Behavioral Twins for Safe Connected Cars*. In Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, pages 144–153, Copenhagen Denmark, 2018. (Cité en page 140.)

[Cheng 2019] Gordon Cheng, Emmanuel Dean-Leon, Florian Bergner, Julio Rogelio Guadarrama Olvera, Quentin Leboutet et Philipp Mittendorfer. *A Comprehensive Realization of Robot Skin : Sensors, Sensing, Control, and Applications*. Proceedings of the IEEE, vol. 107, no. 10, pages 2034–2051, 2019. (Cité en page 10.)

[Chinello 2011] Francesco Chinello, Stefano Scheggi, Fabio Morbidi et Domenico Prattichizzo. *KUKA Control Toolbox*. IEEE Robotics & Automation Magazine, vol. 18, no. 4, pages 69–79, 2011. (Cité en page 17.)

[Chishiro 2009] Hiroyuki Chishiro, Yuji Fujita, Akira Takeda, Yuta Kojima, Kenji Funaoka, Shinpei Kato et Nobuyuki Yamasaki. *Extended RT-Component Framework for RT-Middleware*. In IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, pages 161–168, Tokyo, Japan, 2009. (Cité en page 27.)

[Chitic 2018] Stefan-Gabriel Chitic. *Middleware and Programming Models for Multi-Robot Systems*. PhD thesis, Université de Lyon, Lyon, 2018. (Cité en pages 24, 26, 27, 29, 35 et 36.)

[Claraco 2010] Jose Luis Blanco Claraco. *Development of Scientific Applications with the Mobile Robot Programming Toolkit*. Rapport technique, Machine Perception and Intelligent Robotics Laboratory, University of Malaga, 2010. (Cité en page 33.)

[Company 2021] Shadow Robot Company. *Shadow Teleoperation System*. Technical specification, Shadow Robot Company, 2021. (Cité en page 17.)

[Computing 2012] Twin Oaks Computing. *What Can DDS Do for Android ?* Technical report, Twin Oaks Computing, Inc, 2012. (Cité en page 43.)

[Côté 2007] Carle Côté, Dominic Létourneau, Clément Raïevsky, Yannick Brosseau et François Michaud. *Using MARIE for Mobile Robot Component Development and Integration*. In Davide Brugali, editeur, Software Engineering for Experimental Robotics, volume 30, pages 211–230. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. (Cité en page 32.)

[Cousins 2010] Steve Cousins, Brian Gerkey, Ken Conley et Willow Garage. *Sharing Software with ROS [ROS Topics]*. IEEE Robotics & Automation Magazine, vol. 17, no. 2, pages 12–14, 2010. (Cité en page 33.)

[de Gier 2015] M R de Gier. *Control of a Robotic Arm : Application to on-Surface 3D-printing*. Master thesis, Delft University of Technology, 2015. (Cité en page 17.)

[De Santis 2008] Agostino De Santis, Bruno Siciliano, Alessandro De Luca et Antonio Bicchi. *An Atlas of Physical Human–Robot Interaction*. Mechanism and Machine Theory, vol. 43, no. 3, pages 253–270, 2008. (Cité en page 10.)

[Deng 2021] Min Deng, Carol C. Menassa et Vineet R. Kamat. *From BIM to Digital Twins : A Systematic Review of the Evolution of Intelligent Building Representations in the AEC-FM Industry*. Journal of Information Technology in Construction, vol. 26, pages 58–83, 2021. (Cité en page 140.)

[El Zaatari 2019] Shirine El Zaatari, Mohamed Marei, Weidong Li et Zahid Usman. *Cobot Programming for Collaborative Industrial Tasks : An Overview*. Robotics and Autonomous Systems, vol. 116, pages 162–180, 2019. (Cité en page 7.)

[Elkady 2012] Ayssam Elkady et Tarek Sobh. *Robotics Middleware : A Comprehensive Literature Survey and Attribute-Based Bibliography*. Journal of Robotics, vol. 2012, pages 1–15, 2012. (Cité en page 26.)

[Endo 2004] Y. Endo, D.C. MacKenzie et R.C. Arkin. *Usability Evaluation of High-Level User Assistance for Robot Mission Specification*. IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews), vol. 34, no. 2, pages 168–180, 2004. (Cité en page 31.)

[EPSG 2016] EPSG. *The 5 Major Technologies - 3rd Edition*. Industrial Ethernet Facts - Ethernet Powerlink Standardization Group, 2016. (Cité en pages 47 et 55.)

[Escande 2014] Adrien Escande, Nicolas Mansard et Pierre-Brice Wieber. *Hierarchical Quadratic Programming : Fast Online Humanoid-Robot Motion Generation*. The International Journal of Robotics Research, vol. 33, no. 7, pages 1006–1028, 2014. (Cité en page 132.)

[Fernandez 2006] J.A. Fernandez, C. Galindo et J. Gonzalez. *Integrating Heterogeneous Robotic Software*. In MELECON IEEE Mediterranean Electrotechnical Conference, pages 433–436, Benalmadena, Spain, 2006. (Cité en pages 27 et 32.)

[Ferreau 2007] Hans Joachim Ferreau, Peter Ortner, Peter Langthaler, Luigi del Re et Moritz Diehl. *Predictive Control of a Real-World Diesel Engine Using an Extended Online Active Set Strategy*. Annual Reviews in Control, vol. 31, no. 2, pages 293–301, 2007. (Cité en page 115.)

[Ferreau 2014] Hans Joachim Ferreau, Christian Kirches, Andreas Potschka, Hans Georg Bock et Moritz Diehl. *qpOASES : A Parametric Active-Set Algorithm for Quadratic Programming*. Mathematical Programming Computation, vol. 6, no. 4, pages 327–363, 2014. (Cité en page 115.)

[Ferreau 2017] Hans Joachim Ferreau. *qpOASES User's Manual*, 2017. (Cité en page 115.)

[Fischer 2019] H. Fischer, M. Vulliez, P. Laguillaumie, P. Vulliez et J. P. Gazeau. *RTRobMultiAxisControl : A Framework for Real-Time Multiaxis and Multirobot Control*. IEEE Transactions on Automation Science and Engineering, vol. 16, no. 3, pages 1205–1217, 2019. (Cité en pages 44 et 68.)

[Foundation 2021] OPC Foundation. *OPC UAcademics Lecture*, 2021. (Cité en pages 53 et 56.)

[Fritsche 2015] Lars Fritsche, Felix Unverzag, Jan Peters et Roberto Calandra. *First-Person Tele-Operation of a Humanoid Robot*. In IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids), pages 997–1002, Seoul, South Korea, 2015. (Cité en page 17.)

[Fryman 2012] Jeff Fryman et Björn Matthias. *Safety of Industrial Robots : From Conventional to Collaborative Applications*. 2012. (Cité en page 6.)

[Fuller 2020] Aidan Fuller, Zhong Fan, Charles Day et Chris Barlow. *Digital Twin : Enabling Technologies, Challenges and Open Research*. IEEE Access, vol. 8, pages 108952–108971, 2020. (Cité en pages 142 et 145.)

[Fuseiller 2019] Guillaume Fuseiller. *Perception et Génération de Trajectoires En Cobotique Dans Des Environnements Dynamiques*. PhD thesis, Université de Limoges, Limoges, 2019. (Cité en page 10.)

[Galluzzo 2010] Thomas Galluzzo et Danny Kent. *The OpenJAUS Approach to Designing and Implementinf the New SAE JAUS Standards*. In AUVSI Unmanned Systems Conference, 2010. (Cité en page 33.)

[Garcia 2013] A. Sanchez Garcia, E. Estevez, J. Gomez Ortega et J. Gamez Garcia. *Component-Based Modelling for Generating Robotic Arm Applications Running under OROCOS Middleware*. In IEEE International Conference on Systems, Man, and Cybernetics, pages 3633–3638, Manchester, 2013. (Cité en page 37.)

[Garg 2021] Gaurav Garg, Vladimir Kuts et Gholamreza Anbarjafari. *Digital Twin for FANUC Robots : Industrial Robot Programming and Simulation Using Virtual Reality*. Sustainability, vol. 13, no. 18, page 10336, 2021. (Cité en page 18.)

[Gastebois 2017] Jérémy Gastebois. *Contribution à la commande temps réel des robots marcheurs. Application aux stratégies d'évitement des chutes*. PhD thesis, Université de Poitiers, Poitiers, 2017. (Cité en page 69.)

[Gerkey 2002] B.P. Gerkey et M.J. Mataric. *Sold !: Auction Methods for Multirobot Coordination*. IEEE Transactions on Robotics and Automation, vol. 18, no. 5, pages 758–768, 2002. (Cité en page 46.)

[Glaessgen 2012] Edward Glaessgen et David Stargel. *The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles*. In 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference&lt ;BR&gt ;20th AIAA/ASME/AHS Adaptive Structures Conference&lt ;BR&gt ;14th AIAA, Honolulu, Hawaii, 2012. American Institute of Aeronautics and Astronautics. (Cité en page 143.)

[GmbH 2018] Siemens Healthcare GmbH. *From Digital Twin to Improved Patient Experience-Case Study*, 2018. (Cité en page 142.)

[Gotha 2011] Nils Gotha, Klaus Bernzen, Wilfried Plaß, Joachim Unfried, Martin Schrott, Roland Schaumburg, Jan Braun, Alfred Möltner, Ryszard Bochniak, Djafar Hadiouche, Juergen Hipp, Harald Buchgeher, Candido Ferrio, Josep Lario, Yoshikazu Tachibana, Klas Hellmann, Jan Kosa, Burkhard Werner, Wolfgang Fien, Willi Gagsteiger, Hilmar Panzer, Edwin Schwellinger, Lutz Augenstein et Heiko Berner. *Function Blocks for Motion Control*. page 141, 2011. (Cité en page 61.)

[Gregorio 2020] Jean-Loup Gregorio. *Contribution à la définition d'un jumeau numérique pour la maîtrise de la qualité géométrique des structures aéronautiques lors de leurs processus d'assemblage*. PhD thesis, Université Paris-Saclay, 2020. (Cité en page 142.)

[Grieves 2014] Michael Grieves. *Digital Twin : Manufacturing Excellence through Virtual Factory Replication*, 2014. (Cité en page 142.)

[Haidegger 2013] Tamás Haidegger, Marcos Barreto, Paulo Gonçalves, Maki K. Habib, Sampath Kumar Veera Ragavan, Howard Li, Alberto Vaccarella, Roberta Perrone et Edson Prestes. *Applied Ontologies and Standards for Service Robots*. Robotics and Autonomous Systems, vol. 61, no. 11, pages 1215–1223, 2013. (Cité en page 5.)

[Harper 2019] K. Eric Harper, Somayeh Malakuti et Christopher Ganz. *Digital Twin Architecture and Standards*. 2019. (Cité en pages 142, 143 et 144.)

[He 2019] Rui He, Guoming Chen, Che Dong, Shufeng Sun et Xiaoyu Shen. *Data-Driven Digital Twin Technology for Optimized Control in Process Systems*. ISA Transactions, vol. 95, pages 221–234, 2019. (Cité en page 142.)

[Heckel 2006] Frederick Heckel, Tim Blakely, Michael Dixon, Chris Wilson et William D Smart. *The WURDE Robotics Middleware and RIDE Multi-Robot Tele-Operation Interface*. In AAAI Mobile Robotics Workshop, 2006. (Cité en page 31.)

[Henning 2004] Michi Henning et Mark Spruiell. *Distributed Programming with Ice*. Rapport technique, ZeroC, Inc, 2004. (Cité en page 39.)

[Henning 2008] Michi Henning. *The Rise and Fall of CORBA*. Communications of the ACM, vol. 51, no. 8, pages 52–57, 2008. (Cité en page 39.)

[Hoffmann 2016] Alwin Hoffmann, Alexander Poeppel, Andreas Schierl et Wolfgang Reif. *Environment-Aware Proximity Detection with Capacitive Sensors for Human-Robot-*

Interaction. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 145–150, Daejeon, South Korea, 2016. (Cité en page 10.)

[Hoshi 2006] T. Hoshi et H. Shinoda. *Robot Skin Based on Touch-Area-Sensitive Tactile Element*. In Proceedings IEEE International Conference on Robotics and Automation, 2006. ICRA 2006., pages 3463–3468, Orlando, FL, USA, 2006. (Cité en page 7.)

[Ikuta 2003] Koji Ikuta, Hideki Ishii et Makoto Nokata. *Safety Evaluation Method of Design and Control for Human-Care Robots*. The International Journal of Robotics Research, vol. 22, no. 5, pages 281–297, 2003. (Cité en page 9.)

[Jakel 2011] R. Jakel, P. Meissner, Sven R. Schmidt-Rohr et R. Dillmann. *Distributed Generalization of Learned Planning Models in Robot Programming by Demonstration*. In IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 4633–4638, San Francisco, CA, 2011. (Cité en page 8.)

[Jang 2010] Choulsoo Jang. *OPRoS : A New Component-Based Robot Software Platform*. ETRI Journal, vol. 32, no. 5, pages 646–656, 2010. (Cité en pages 27, 33 et 35.)

[Järvinen 1995] Jari Järvinen et Waldemar Karwowski. *Analysis of Self-Reported Accidents Attributed to Advanced Manufacturing Systems*. International Journal of Human Factors in Manufacturing, vol. 5, no. 3, pages 251–266, 1995. (Cité en page 5.)

[Ji 2007] Meng Ji et Magnus Egerstedt. *Distributed Coordination Control of Multiagent Systems While Preserving Connectedness*. IEEE Transactions on Robotics, vol. 23, no. 4, pages 693–703, 2007. (Cité en page 8.)

[Johns 2008] Kyle Johns et Trevor Taylor. Professional Microsoft robotics developer studio. Wrox Professional Guides. Wiley Pub, Indianapolis, IN, 2008. (Cité en page 32.)

[Joseph 2020] Lucas Joseph, Joshua K. Pickard, Vincent Padois et David Daney. *Online Velocity Constraint Adaptation for Safe and Efficient Human-Robot Workspace Sharing*. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 11045–11051, Las Vegas, NV, USA, 2020. (Cité en page 138.)

[Kagermann 2013] Henning Kagermann, Wolfgang Wahlster et Johannes Helbig. *Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0 – Securing the Future of German Manufacturing Industry*. Technical report, 2013. (Cité en pages 50 et 51.)

[Katti 2020] Badarinath Katti. *Ontology-Based Approach to Decentralized Production Control in the Context of Cloud Manufacturing Execution Systems*. PhD thesis, 2020. (Cité en page 53.)

[Kramer 2007] James Kramer et Matthias Scheutz. *Development Environments for Autonomous Mobile Robots : A Survey*. Autonomous Robots, vol. 22, no. 2, pages 101–132, 2007. (Cité en page 26.)

[Kranz 2006] Matthias Kranz, Radu Bogdan Rusu, Alexis Maldonado, Michael Beetz et Albrecht Schmidt. *A Player/Stage System for Context-Aware Intelligent Environments*. In Proceedings of UbiSys, 2006. (Cité en pages 27 et 32.)

[Kröger 2010] Torsten Kröger. On-Line Trajectory Generation in Robotic Systems, volume 58 of *Springer Tracts in Advanced Robotics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. (Cité en page 89.)

[Kröger 2011] Torsten Kröger. *Opening the Door to New Sensor-Based Robot Applications—The Reflexxes Motion Libraries*. In IEEE International Conference on Robotics and Automation, pages 1–4, Shanghai, China, 2011. (Cité en pages 89 et 90.)

[Kruger 2006] Daniel Kruger, Ingo van Lil, Niko Sunderhauf, Robert Baumgartl et Peter Protzel. *Using and Extending the Miro Middleware for Autonomous Mobile Robots*. In Towards Autonomous Robotic Systems (TAROS), Guildford, 2006. (Cité en pages 27 et 32.)

[Kwak 2006] Jun-young Kwak, Ji Young Yoon et R.H. Shinn. *An Intelligent Robot Architecture Based on Robot Mark-up Languages*. In IEEE International Conference on Engineering of Intelligent Systems, pages 1–6, Islamabad, Pakistan, 2006. (Cité en pages 27 et 31.)

[Laconte 2019] Johann Laconte, Christophe Debain, Roland Chapuis, Francois Pomerleau et Romuald Aufrere. *Lambda-Field : A Continuous Counterpart of the Bayesian Occupancy Grid for Risk Assessment*. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 167–172, Macau, China, 2019. (Cité en page 10.)

[Laconte 2021] Johann Laconte, Elie Randriamiarintsoa, Abderrahim Kasmi, Francois Pomerleau, Roland Chapuis, Christophe Debain et Romuald Aufrere. *Dynamic Lambda-Field : A Counterpart of the Bayesian Occupancy Grid for Risk Assessment in Dynamic Environments*. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 4846–4853, Prague, Czech Republic, 2021. (Cité en page 10.)

[Liu 2018] Zheng Liu, Norbert Meyendorf et Nezih Mrad. *The Role of Data Fusion in Predictive Maintenance Using Digital Twin*. In 44TH ANNUAL REVIEW OF PROGRESS IN QUANTITATIVE NONDESTRUCTIVE EVALUATION, VOLUME 37, page 020023, Provo, Utah, USA, 2018. (Cité en page 144.)

[Magnenat 2011] S Magnenat, P Rétornaz, M Bonani, V Longchamp et F Mondada. *ASEBA : A Modular Architecture for Event-Based Control of Complex Robots*. IEEE/ASME Transactions on Mechatronics, vol. 16, no. 2, pages 321–329, 2011. (Cité en page 33.)

[Malakuti 2019] S. Malakuti, J. Schlake, C. Ganz, K. E. Harper et H. Petersen. *Digital Twin : An Enabler for New Business Models*. In VDI Wissensforum GmbH, editeur, Automation 2019, pages 807–820. VDI Verlag, 2019. (Cité en page 145.)

[Maurice 2017] Pauline Maurice, Vincent Padois, Yvan Measson et Philippe Bidaud. *Human-Oriented Design of Collaborative Robots*. International Journal of Industrial Ergonomics, vol. 57, pages 88–102, 2017. (Cité en page 10.)

[Mawson 2019] Victoria Jayne Mawson et Ben Richard Hughes. *The Development of Modelling Tools to Improve Energy Efficiency in Manufacturing Processes and Systems*. Journal of Manufacturing Systems, vol. 51, pages 95–105, 2019. (Cité en page 142.)

[Metta 2006] Giorgio Metta, Paul Fitzpatrick et Lorenzo Natale. *YARP : Yet Another Robot Platform*. International Journal of Advanced Robotic Systems, vol. 3, no. 1, page 8, 2006. (Cité en page 32.)

[Michel 2004] Olivier Michel. *Cyberbotics Ltd. Webots ™ : Professional Mobile Robot Simulation*. International Journal of Advanced Robotic Systems, vol. 1, no. 1, page 5, 2004. (Cité en page 31.)

[Mizera 2019] Camille Mizera. *Développement d'un préhenseur multi-digital marinisé sensible en efforts dans un contexte de télémanipulation*. PhD thesis, Université de Poitiers, 2019. (Cité en page 151.)

[Mohamed 2008] Nader Mohamed, Jameela Al-Jaroodi et Imad Jawhar. *Middleware for Robotics : A Survey*. In IEEE Conference on Robotics, Automation and Mechatronics, pages 736–742, Chengdu, China, 2008. (Cité en pages 26 et 29.)

[Mohamed 2009] Nader Mohamed, Jameela Al-Jaroodi et Imad Jawhar. *A Review of Middleware for Networked Robots*. 2009. (Cité en pages 26 et 29.)

[Montemerlo 2003] Michael Montemerlo, Nicholas Roy et Sebastian Thrun. *Perspectives on Standardization in Mobile Robot Programming : The Carnegie Mellon Navigation (CARMEN) Toolkit*. Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003), 2003. (Cité en page 31.)

[Morilhat 2018] Patrick Morilhat. *Digitalization of Nuclear Power Plants at EDF*. In Energiforsk Annual Nuclear Conference, 2018. (Cité en page 141.)

[Mouad 2014] Mehdi Mouad. *Architecture de COntrôle/COmmande dédiée aux systèmes Distribués Autonomes (ACO^{2}DA) : application à une plate-forme multi-véhicules*. PhD thesis, Université Blaise Pascal, Clermont 2, 2014. (Cité en page 13.)

[Murphy 2000] Robin Murphy. Introduction to AI robotics. Intelligent Robotics and Autonomous Agents. MIT Press, Cambridge, Mass, 2000. (Cité en page 11.)

[Nagamachi 1986] Mitsuo Nagamachi. *Human Factors of Industrial Robots and Robot Safety Management in Japan*. Applied Ergonomics, vol. 17, no. 1, pages 9–18, 1986. (Cité en page 5.)

[Nesnas 2006] Issa A.D. Nesnas, Reid Simmons, Daniel Gaines, Clayton Kunz, Antonio Diaz-Calderon, Tara Estlin, Richard Madison, John Guineau, Michael McHenry, I-Hsiang Shu et David Apfelbaum. *CLARAty : Challenges and Steps toward Reusable Robotic Software*. International Journal of Advanced Robotic Systems, vol. 3, no. 1, page 5, 2006. (Cité en page 31.)

[Nicolson 2016] Edward Nicolson et Yaskawa America. *The Quest for a Single Controller*. page 10, 2016. (Cité en page 98.)

[Nof 1999] Shimon Y. Nof, editeur. Handbook of industrial robotics. John Wiley, New York, 2nd ed édition, 1999. (Cité en page 5.)

[Norman 1986] Donald A. Norman et Tim Shallice. *Attention to Action : Willed and Automatic Control of Behavior*. In Richard J. Davidson, Gary E. Schwartz et David Shapiro, editeurs, Consciousness and Self-Regulation, pages 1–18. Springer US, Boston, MA, 1986. (Cité en page 13.)

[Omarali 2017] Bukeikhan Omarali, Tasbolat Taunyazov, Askhat Bukeyev et Almas Shintemirov. *Real-Time Predictive Control of an UR5 Robotic Arm Through Human Upper Limb Motion Tracking*. In Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction, pages 414–414, Vienna Austria, 2017. ACM. (Cité en page 17.)

[Otani 2018] Kazuya Otani, Karim Bouyarmane et Serena Ivaldi. *Generating Assistive Humanoid Motions for Co-Manipulation Tasks with a Multi-Robot Quadratic Program Controller*. In IEEE International Conference on Robotics and Automation (ICRA), pages 3107–3113, Brisbane, QLD, 2018. (Cité en page 10.)

[Pang 2021] Gaoyang Pang, Geng Yang et Zhibo Pang. *Review of Robot Skin : A Potential Enabler for Safe Collaboration, Immersive Teleoperation, and Affective Interaction of Future Collaborative Robots*. IEEE Transactions on Medical Robotics and Bionics, vol. 3, no. 3, pages 681–700, 2021. (Cité en page 10.)

[Park 2011] Jung-Jun Park, Sami Haddadin, Jae-Bok Song et Alin Albu-Schaffer. *Designing Optimally Safe Robot Surface Properties for Minimizing the Stress Characteristics of Human-Robot Collisions*. In IEEE International Conference on Robotics and Automation, pages 5413–5420, Shanghai, China, 2011. (Cité en page 10.)

[Park 2016] Sungman Park, Yeongtae Jung et Joonbum Bae. *A Tele-Operation Interface with a Motion Capture System and a Haptic Glove*. In 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), pages 544–549, Xian, China, 2016. IEEE. (Cité en page 17.)

[Park 2020] Jaeho Park, Raimarius Delgado et Byoung Wook Choi. *Real-Time Characteristics of ROS 2.0 in Multiagent Robot Systems : An Empirical Study*. IEEE Access, vol. 8, pages 154637–154651, 2020. (Cité en page 45.)

[Parker 1998] L.E. Parker. *ALLIANCE : An Architecture for Fault Tolerant Multirobot Cooperation*. IEEE Transactions on Robotics and Automation, vol. 14, no. 2, pages 220–240, 1998. (Cité en page 46.)

[Passama 2014] Robin Passama, David Andreu, Didier Crestani et Karen Godary-Dejean. *Architectures de contrôle pour la robotique - Approches et tendances*. Robotique, 2014. (Cité en page 12.)

[PLCopen 2018] PLCopen. *Safety Software, Technical Specification, Version 2.0. Technical Committee 5*, 2018. (Cité en pages 61 et 62.)

[Ponsà Cobas 2020] Sergi Ponsà Cobas. *Strategies for Remote Control and Teleoperation of a UR Robot*. Master thesis, Escola Tècnica Superior d'Enginyeria Industrial de Barcelona, 2020. (Cité en page 17.)

[Pope 1998] Alan Pope. The CORBA reference guide : Understanding the common object request broker architecture. Addison-Wesley, Reading, Mass, 1998. (Cité en pages 21 et 22.)

[Portugal 2021] David Portugal, André G Araújo et Micael S Couceiro. *Improving the Robustness of a Service Robot for Continuous Indoor Monitoring : An Incremental Approach*. International Journal of Advanced Robotic Systems, vol. 18, no. 3, page 172988142110121, 2021. (Cité en page 8.)

[Qi 2018] Qinglin Qi et Fei Tao. *Digital Twin and Big Data Towards Smart Manufacturing and Industry 4.0 : 360 Degree Comparison*. IEEE Access, vol. 6, pages 3585–3593, 2018. (Cité en page 142.)

[Qilin 2010] Li Qilin et Zhou Mintian. *The State of the Art in Middleware*. In International Forum on Information Technology and Applications, pages 83–85, Kunming, China, 2010. IEEE. (Cité en page 21.)

[Ramos 1996] C. Ramos. *A Holonic Approach for Task Scheduling in Manufacturing Systems*. In Proceedings of IEEE International Conference on Robotics and Automation, volume 3, pages 2511–2516, Minneapolis, MN, USA, 1996. (Cité en page 46.)

[Ravn 2014] O. Ravn, N. A. Andersen et T. T. Andersen. *UR10 Performance Analysis*. Rapport technique, Technical University of Denmark, Department of Electrical Engineering, 2014. (Cité en page 17.)

[Riccoboni 2022] Jean-Baptiste Riccoboni. *Modélisation dynamique d'arborescence - application à la robotique et à l'étude du mouvement humain*. PhD thesis, Université de Poitiers, Poitiers, 2022. (Cité en page 165.)

[Rodríguez 2016] Enrique Rodríguez, José Luis Blanco, José Luis Torres, Jose Carlos Moreno, Antonio Giménez et José Luis Guzmán. *A ROS Reactive Navigation System for Ground Vehicles Based on Tp-Space Transformations*. In Actas de Las XXXVII Jornadas de Automática 7, 8 y 9 de Septiembre de 2016, Madrid, pages 1213–1220. Universidade da Coruña, Servizo de Publicacións, 2016. (Cité en page 33.)

[Russell 2010] Stuart J. Russell, Peter Norvig et Ernest Davis. Artificial intelligence : A modern approach. Prentice Hall Series in Artificial Intelligence. Prentice Hall, Upper Saddle River, 3rd ed édition, 2010. (Cité en page 12.)

[Sandholm 1993] Tuomas Sandholm. *An Implementation of the Contract Net Protocol Based on Marginal Cost Calculations*. In AAAI Conference on Artificial Intelligence, 1993. (Cité en page 46.)

[Sanfilippo 2015] Filippo Sanfilippo, Lars Ivar Hatledal, Houxiang Zhang, Massimiliano Fago et Kristin Y. Pettersen. *Controlling Kuka Industrial Robots : Flexible Communication Interface JOpenShowVar*. IEEE Robotics & Automation Magazine, vol. 22, no. 4, pages 96–109, 2015. (Cité en page 18.)

[Schlegel 2012] Christian Schlegel, Andreas Steck et Alex Lotz. *Robotic Software Systems : From Code-Driven to Model-Driven Software Development*. In Ashish Dutta, editeur, Robotic Systems - Applications, Control and Programming. InTech, 2012. (Cité en page 32.)

[Schmidt 2005] M.-T. Schmidt, B. Hutchison, P. Lambros et R. Phippen. *The Enterprise Service Bus : Making Service-Oriented Architecture Real*. IBM Systems Journal, vol. 44, no. 4, pages 781–797, 2005. (Cité en page 24.)

[Schopfer 2010] Matthias Schopfer, Florian Schmidt, Michael Pardowitz et Helge Ritter. *Open Source Real-Time Control Software for the Kuka Light Weight Robot*. In 8th World Congress on Intelligent Control and Automation, pages 444–449, Jinan, China, 2010. (Cité en page 18.)

[Siciliano 2008] Bruno Siciliano et Oussama Khatib. Springer handbook of robotics. Springer, Berlin, 2008. (Cité en page 11.)

[Soe 2017] Ralf-Martin Soe. *FINEST Twins : Platform for Cross-Border Smart City Solutions*. In Proceedings of the 18th Annual International Conference on Digital Government Research, pages 352–357, Staten Island NY USA, 2017. ACM. (Cité en page 140.)

[Stark 2019] Rainer Stark et Thomas Damerau. *Digital Twin*. In The International Academy for Production Engineering, Sami Chatti et Tullio Tolio, editeurs, CIRP Encyclopedia of Production Engineering, pages 1–8. Springer Berlin Heidelberg, Berlin, Heidelberg, 2019. (Cité en page 144.)

[Syntec-Ingénierie 2021] Syntec-Ingénierie. *Le jumeau numérique au service du changement climatique*, 2021. (Cité en pages 145 et 146.)

[Tolley 2014] Michael T. Tolley, Robert F. Shepherd, Bobak Mosadegh, Kevin C. Galloway, Michael Wehner, Michael Karpelson, Robert J. Wood et George M. Whitesides. *A Resilient, Untethered Soft Robot.* Soft Robotics, vol. 1, no. 3, pages 213–223, 2014. (Cité en page 8.)

[Tonietti 2005] G. Tonietti, R. Schiavi et A. Bicchi. *Design and Control of a Variable Stiffness Actuator for Safe and Fast Physical Human/Robot Interaction.* In Proceedings of the IEEE International Conference on Robotics and Automation, pages 526–531, Barcelona, Spain, 2005. (Cité en page 10.)

[Ulmen 2010] John Ulmen et Mark Cutkosky. *A Robust, Low-Cost and Low-Noise Artificial Skin for Human-Friendly Robots.* In IEEE International Conference on Robotics and Automation, pages 4836–4841, Anchorage, AK, 2010. (Cité en page 10.)

[UNECE 2022] UNECE. *Codes for Trade | UNECE.* https ://unece.org/trade/cefact/UNLOCODE-Download, 2022. (Cité en page 87.)

[Unnikrishnan 2017] Nishant Unnikrishnan, Kevin Hull et Edward Nicolson. *A Review of Challenges in Integrating Robot and Motion Control Into a Single System.* In Volume 14 : Emerging Technologies ; Materials : Genetics to Structures ; Safety Engineering and Risk Analysis, page V014T07A017, Tampa, Florida, USA, 2017. American Society of Mechanical Engineers. (Cité en pages 98 et 99.)

[Valle 2013] Daniela Valle, Emmanuel Nuno, Luis Basanez et Nancy Arana-Daniel. *Consensus of Networks of Nonidentical Robots with Flexible Joints, Variable Time-Delays and Immeasurable Velocities.* In IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 5878–5883, Tokyo, 2013. (Cité en page 26.)

[van der Hoorn 2022] Gijs van der Hoorn. *Fanuc_driver_exp.* Technische Universiteit Delft, 2022. (Cité en page 18.)

[Vanderborght 2013] B. Vanderborght, A. Albu-Schaeffer, A. Bicchi, E. Burdet, D.G. Caldwell, R. Carloni, M. Catalano, O. Eiberger, W. Friedl, G. Ganesh, M. Garabini, M. Grebenstein, G. Grioli, S. Haddadin, H. Hoppner, A. Jafari, M. Laffranchi, D. Lefeber, F. Petit, S. Stramigioli, N. Tsagarakis, M. Van Damme, R. Van Ham, L.C. Visser et S. Wolf. *Variable Impedance Actuators : A Review.* Robotics and Autonomous Systems, vol. 61, no. 12, pages 1601–1614, 2013. (Cité en page 10.)

[Varillon 2021] Benoit Varillon, Jean-Baptiste Chaudron, David Doose et Charles Lesire. *Corail, ROS2 temps réel.* In ROSConFr 21, France, 2021. (Cité en page 45.)

[Villani 2018] Valeria Villani, Fabio Pini, Francesco Leali et Cristian Secchi. *Survey on Human–Robot Collaboration in Industrial Settings : Safety, Intuitive Interfaces and Applications.* Mechatronics, vol. 55, pages 248–266, 2018. (Cité en page 6.)

[Vrabič 2018] Rok Vrabič, John Ahmet Erkoyuncu, Peter Butala et Rajkumar Roy. *Digital Twins : Understanding the Added Value of Integrated Models for through-Life Engineering Services.* Procedia Manufacturing, vol. 16, pages 139–146, 2018. (Cité en page 144.)

[Vulliez 2018] Philippe Vulliez, Jean-Pierre Gazeau et Pierre Laguillaumie. *RTRobMultiAxisControl*, 2018. (Cité en page 68.)

[Vulliez 2019] Philippe Vulliez. *Conception Mécatronique d'une Main Robotique En Vue de Tâches de Manipulation Fine. Développement et Implémentation Du Framework RTRobMultiAxisControl.* PhD thesis, Université de Poitiers, Poitiers, 2019. (Cité en page 68.)

[Wieczorek 2020] F.H. Wieczorek et J. Zhang. *Universal Teleoperation ROS Interface for Robotic Manipulators*. Rapport technique Bachelor thesis, Universität Hamburg Fakultät für Mathematik, Informatik und Naturwissenschaften and Universität Hamburg Fachbereich Informatik, 2020. (Cité en page 17.)

[Yadav 2021] Vaibhav Yadav, Vivek Agarwal, Andrei V. Gribok, Ross D. Hays, Adam J. Pluth, Christopher S. Ritter et Hongbin Zhang. *Technical Challenges and Gaps in Digital-Twin-Enabling Technologies for Nuclear Reactor Applications*. Letter Report U.S. Nuclear Regulatory Commission TLR/RES-DE-REB-2021-17, Idaho National Laboratory, 2021. (Cité en page 141.)

[Yim 2007] Mark Yim, Wei-min Shen, Behnam Salemi, Daniela Rus, Mark Moll, Hod Lipson, Eric Klavins et Gregory Chirikjian. *Modular Self-Reconfigurable Robot Systems [Grand Challenges of Robotics]*. IEEE Robotics & Automation Magazine, vol. 14, no. 1, pages 43–52, 2007. (Cité en page 8.)

[Yu 2022] Wei Yu, Panos Patros, Brent Young, Elsa Klinac et Timothy Gordon Walmsley. *Energy Digital Twin Technology for Industrial Energy Management : Classification, Challenges and Future*. Renewable and Sustainable Energy Reviews, vol. 161, page 112407, 2022. (Cité en page 141.)

[Zhang 2010] W. J. Zhang et Y. Lin. *On the Principle of Design of Resilient Systems – Application to Enterprise Information Systems*. Enterprise Information Systems, vol. 4, no. 2, pages 99–110, 2010. (Cité en page 9.)

[Zhang 2017] Tan Zhang, Wenjun Zhang et Madan Gupta. *Resilient Robots : Concept, Review, and Future Directions*. Robotics, vol. 6, no. 4, page 22, 2017. (Cité en page 8.)

Résumé

La conception de machines de production doit répondre à de nouvelles exigences : intégration dans un système d'information étendu, capacité d'adaptation et de prise de décision décentralisée et virtualisation. Le logiciel de pilotage des machines devenant de plus en plus complexe, de nombreux intergiciels ont été conçus pour en simplifier le développement. La première partie de ce mémoire présente les principaux intergiciels orientés robotique et en identifie les limites. Dans le cas de la conception de systèmes mécatroniques qui doivent réagir rapidement et de façon robuste à des perturbations de leur environnement, nous avons développé une solution logicielle générique de contrôle multiaxe basée sur les caractéristiques suivantes :

— respect des standards IEC 61131 et PLCopen Motion Control,
— usage des protocoles de communication Ethernet temps réel,
— implémentation possible sur une variété de plateformes (PC, PLC, microcontrôleurs),
— architecture avec une distribution modulable des composants logiciels et des boucles de contrôle de bas niveau associées en fonction des ressources matérielles disponibles,
— intégration au plus bas niveau d'algorithmes de programmation quadratique,
— lien haute-fidélité entre machine réelle et machine virtuelle exploitant une simulation intégrée.

Le mémoire présente la démarche suivie pour concevoir de façon générique des logiciels de contrôle de mécanismes et de robots et illustre cette approche avec le développement logiciel d'un préhenseur multidigital sous-actionné et la création de jumeaux numériques connectés.

Mots clés : contrôle-commande multiaxe, middleware robotique, automates programmables industriels, développement logiciel, jumeau numérique

Software development for reactive multi-axis control : from low-level control to connected digital twin

Abstract

The design of manufacturing machines must meet new requirements : integration into an extended information system, adaptability, decentralized decision-making, and virtualization. As the control software for machines becomes increasingly complex, numerous middleware solutions have been designed to simplify their development. The first part of this thesis presents the main robotics-oriented middleware solutions and identifies their limitations. In the case of designing mechatronic systems that need to react quickly and robustly to disturbances in their environment, we have developed a generic software solution for multi-axis control based on the following characteristics :

- Compliance with IEC 61131 and PLCopen Motion Control standards
- Use of real-time Ethernet communication protocols
- Possible implementation on a variety of platforms (PC, PLC, microcontrollers)
- Architecture with modular distribution of software components and associated low-level control loops based on available hardware resources
- Integration of quadratic programming algorithms at the lowest level
- High-fidelity link between the real machine and a virtual machine exploiting integrated simulation.

The thesis presents how to generically design control software for mechanisms and robots and illustrates this approach with the software development of an underactuated multidigital gripper and the creation of connected digital twins.

Keywords : Multi-axis control, Robotic middleware, Programmable logical controllers (PLCs), Software development, Digital twin