Hes·so
Haute Ecole Spécialisée
de Suisse occidentale

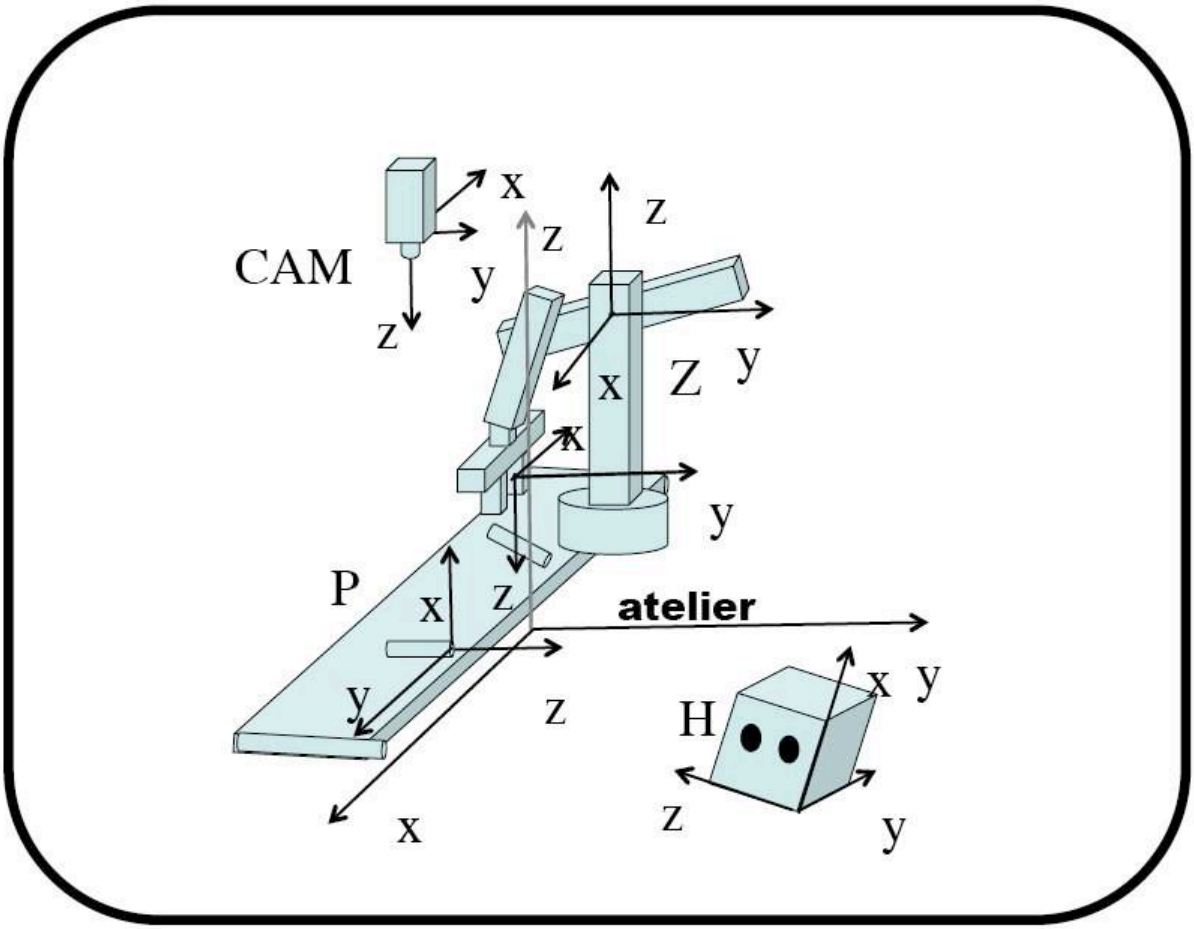
heig-vd
HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Institut d'
Automatisation
Industrielle
LaRA
Laboratoire de Robotique et Automatisation

COMMANDE DES ROBOTS ET DES SYSTEMES AUTOMATISES

(Partie **3** du cours "Robotique et Automatisation")

Jean-Daniel Dessimoz



Yverdon-les-Bains, 1 novembre 2017

COMMANDE DES ROBOTS ET DES SYSTEMES AUTOMATISES

TABLE DES MATIÈRES

COMMANDE DES ROBOTS ET DES SYSTEMES AUTOMATISES

RESUME

Les systèmes de commandes sont de plus en plus organisés hiérarchiquement, que ce soit à l'échelle de tout un atelier de fabrication, ou à celle beaucoup plus petite d'un équipement de production. C'est en particulier le cas pour la commande des robots industriels.

Suivant le niveau à laquelle se trouve une boucle de réglage, des caractéristiques différentes apparaissent (temps de réaction, qualité d'abstraction des grandeurs, interactions avec l'homme...). En conséquence, il y faut des outils spécifiques. Il vaut la peine d'étudier la commande à quelques uns de ces niveaux types, considérés isolément.

Aux niveaux les plus élevés, la commande de systèmes se fait typiquement dans le contexte de programmes. Différentes approches sont adoptées. Nous y trouvons les langages généraux, tels que C, Pascal, ou Ada, permettant en particulier de programmer des ordinateurs industriels; mais il existe aussi, plus spécialisés, les codes à relais ou le langage Grafcet pour les automates programmables; pour les robots, les langages tels que V+(version la plus récente de VAL), RAPID (ABB) ou Comau offrent, en plus d'instructions générales, des mots-clefs adaptés à la gestion de trajectoires et de positions dans l'espace. C'est aussi le cas du langage « Piaget » que nous avons inventé pour nos robots mobiles autonomes.

A un niveau intermédiaire, les robots requièrent une forme particulière de commande afin d'assurer la coordination étroite d'articulations diverses, de façon à suivre des trajectoires dans l'espace avec des caractéristiques dynamiques et de précision sévères. Il s'agit ici non seulement de commander individuellement des systèmes de réglage avals, mais il faut encore assurer un synchronisme entre eux. La coordination est d'autant plus difficile à garantir que tant les systèmes eux-mêmes que les influences externes sur eux sont fort disparates.

Enfin, cette section du cours s'intéresse aux servocommandes, qui servent généralement ici à assurer en permanence, pour chaque articulation individuellement, le positionnement demandé par les étages amonts. Autrefois exclusivement de type analogique, les servocommandes sont maintenant presque toujours numériques. Cependant les algorithmes implémentés ont peu changé, car la théorie dans ce domaine nouveau n'a guère dépassé le stade du mimétisme: sous forme numérique, ce sont pour beaucoup les anciennes techniques que l'on reprend.

3.1 NOTION DE COMMANDE HIERARCHISEE

La commande de systèmes automatisés suit deux types d'évolutions complémentaires. D'une part, les méthodes s'affinent, pour une boucle de commande donnée (voir par exemple fig. 3.1.1). Ces techniques ressortent du réglage automatique. Mais d'autre part, la commande de systèmes complexes font un usage toujours accru de structures hiérarchisées, où de nombreuses boucles de commandes interviennent, à différents niveaux d'abstraction. La puissance de ces systèmes de commande - et parfois la difficulté à les gérer - provient non plus d'une boucle en particulier, mais de leur synergie et de leur interactions mutuelles. Si l'automaticien est souvent embarrassé pour prédire le comportement d'une hiérarchie de systèmes de réglage, il est en revanche parfois surpris de la relative simplicité de certaines boucles considérées isolément.

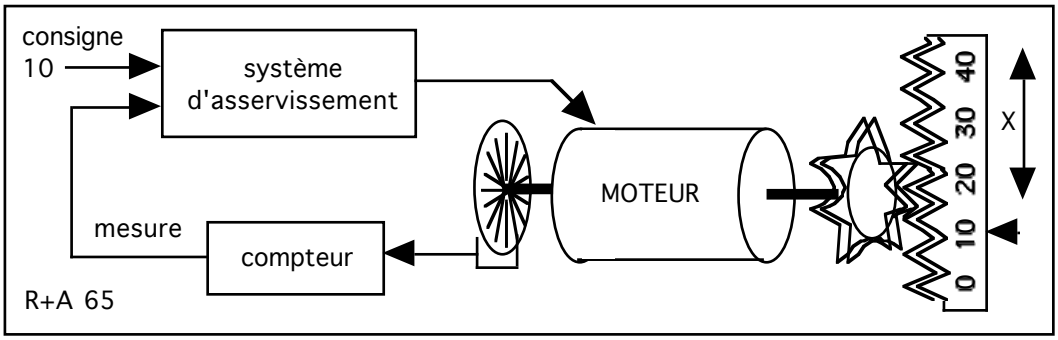


Fig. 3.1.1 Boucle de réglage pour la commande d'un axe.

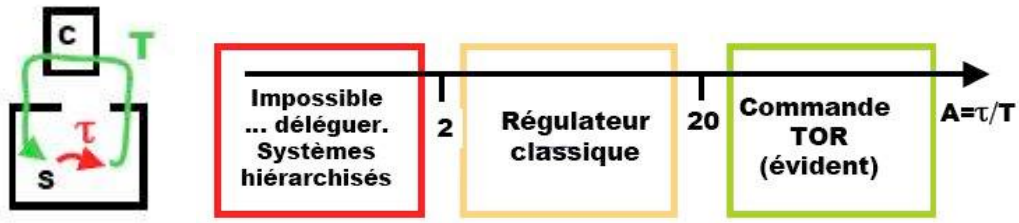


Fig. 3.1.1b On observe que pour des commandes, C, rapides avec retards faibles (T petit, donc vitesse de réaction, ou agilité, grande, A_C=**1/** T), les solutions simples sont appropriées. Lorsqu'au contraire, T avoisine ou dépasse la constante de temps caractéristique , **τ**, du système à commander, S, des modes de régulation plus évolués doivent s'envisager. L'agilité *relative*, A_r, est définie comme le rapport de la seconde grandeur sur la première (A_r= A_C / A_S =**τ**/T ; voir éventuellement plus de détails en §3.4 Servocommandes).

Voyons brièvement deux types de hiérarchies de commande maintenant courantes. La première contrôle des équipements divers, coopérant dans la même cellule de fabrication, dans des îlots ou même à l'échelle de tout un atelier. La deuxième est moins connue, et se découvre à l'intérieur des

commandes de robots industriels. Ensuite, une brève présentation de la communication par réseaux de données sera faite.

3.1.1 CONTROLE D'ATELIER ET DE CELLULES DE FABRICATION

On s'achemine maintenant vers l'automatisation globale des usines de fabrication. L'évolution se fait en même temps à plusieurs échelles. Voyons d'abord celle de tout un atelier, puis nous aborderons celle d'un cellule de fabrication.

A NIVEAU ATELIER

L'automatisation au niveau global de l'atelier pose des problèmes très sérieux. Pour les résoudre, la stratégie du "diviser pour régner" est la seule issue. Partant du centre informatique de l'atelier, l'information est distribuée à un nombre limité de partenaires. Ceux-ci, correspondant à des îlots de production ont à leur tour un nombre limité de cellules, qui elles-mêmes gèrent individuellement leurs machines. Il est clair que l'information remonte aussi des machines jusque vers le centre de l'atelier.

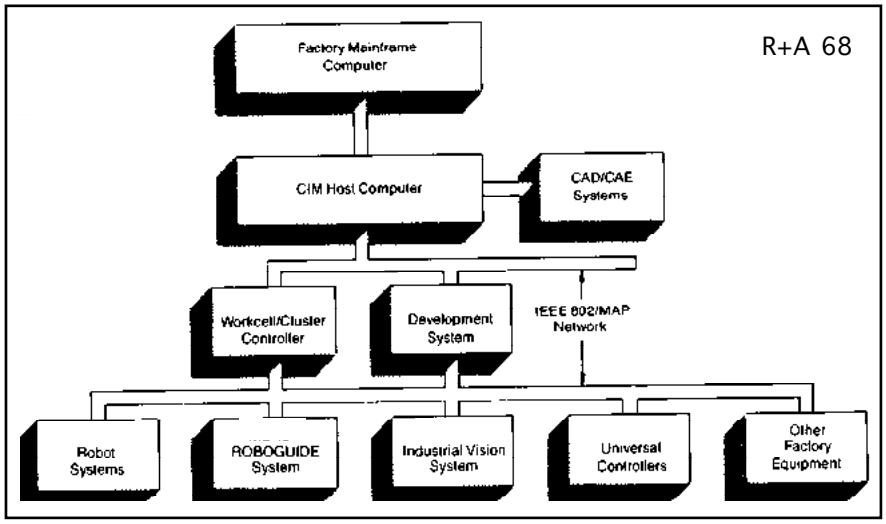


Fig. 3.1.2 Exemple de hiérarchie de commandes au niveau de l'atelier (doc. American Robot/ACC)

La fonction de communication est vitale à ce niveau, et le standard le plus prometteur est celui de MAP, qui en est maintenant à la troisième version de sa définition. MAP suit des normes tant du point de vue physique qu'en ce qui concerne les protocoles d'échanges, jusqu'à un niveau relativement abstrait. Il ressemble au réseau Ethernet, largement répandu, mais s'en distingue en particulier par le fait qu'il garantit un temps de réponse de durée limitée.

B NIVEAU CELLULE DE FABRICATION

A l'intérieur une cellule de fabrication, l'information circule également par des voies hiérarchiques. Cependant, entre les diverses commandes qui y coopèrent, les distances sont courtes, et les intervalles de temps pour synchronisation se réduisent à l'ordre de la seconde. La communication est assurée par des moyens spécifiques entre équipements. Mais de plus en plus, on voit apparaître des

réseaux plus simples. Dans ce contexte, une norme allégée de MAP est aussi en développement: mini-MAP. La tendance touche jusqu'aux instruments, tels qu'un capteur ou une électrovanne. On parle alors de réseaux "de terrain". Une norme international est en préparation. On peut s'attendre, même au niveau le plus bas, à retrouver un sous-ensemble des normes MAP.

A ce niveau de l'atelier de fabrication, les équipements informatiques les plus répandus sont les automates programmables (API), les ordinateurs (OI), et les commandes de robots industriels (RI). Chacun de ces différents types peut être représenté par un organe de contrôle unique, mais très souvent, il s'intègre dans une hiérarchie propre.

Le qualificatif "industriel" que ces organes de commandes partagent indiquent qu'ils sont destinés à travailler en atelier (c'est à dire en milieu riche en perturbations de tous ordres), et à piloter des procédés en temps réel.

Les API sont des machines électroniques, programmables par un personnel peu spécialisé, et destinées à piloter des procédés logiques séquentiels et combinatoires. Typiquement, ils élaborent des signaux binaires de sortie, commandant par exemple des électrovannes pneumatiques, en fonction de signaux binaires d'entrée, tels que délivrés par des interrupteurs ou des barrières lumineuses. Ils s'interconnectent facilement, gardant une structure et une fonction similaires à tous les niveaux.

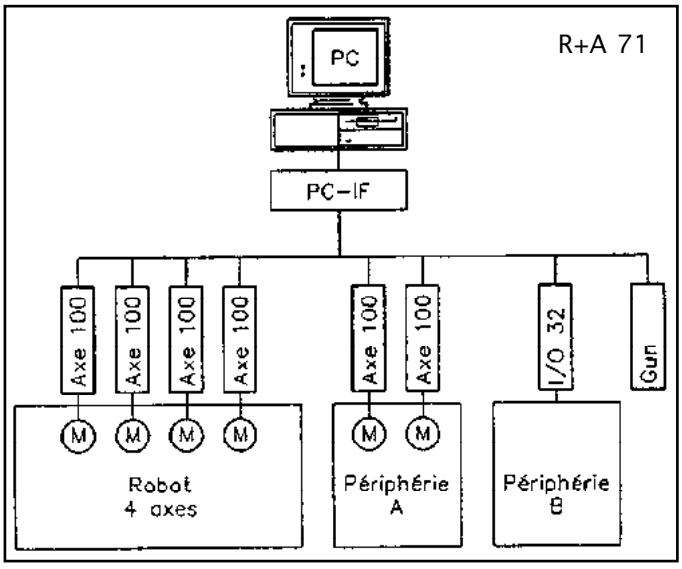


Fig. 3.1.3 Exemple de hiérarchie de commande dans une cellule de fabrication (doc. Demaurex)

Du point de vue fonctionnel, Les OI ne se distinguent guère, en général, des ordinateurs courants. Ils se distinguent par contre des API, par la complexité des procédés qu'ils peuvent gérer. Les entrées/sorties (E/S) ne sont généralement plus simplement binaires, mais se font par nombre ou même par signaux analogiques. Les fonctions liant les sorties aux entrées ne se laissent pas facilement réduire à des systèmes logiques, mais se définissent plutôt par des programmes généralement écrits par des informaticiens. Alors que traditionnellement, on rencontrait souvent là des "mini-ordinateurs", c'est maintenant des systèmes basés sur les microprocesseurs Motorola ou Intel qui abondent (systèmes à bus VME, Multibus, ou IBM-PC...). Les OI, dans ce contexte, pilotent en général à leur tour d'autres commandes, telles que des API ou des servocommandes d'axes mécaniques.

Il existe également toute une gamme de microcontrôleurs et de micro-ordinateurs monochips,, et qui, du point de complexité, se trouvent aux niveaux les plus bas de la hiérarchie des ordinateurs. Ils offrent un rapport intéressant puissance/coût, et sont particulièrement compétitifs lorsqu'il s'agit de commander un processus sans grande interactivité avec l'homme.

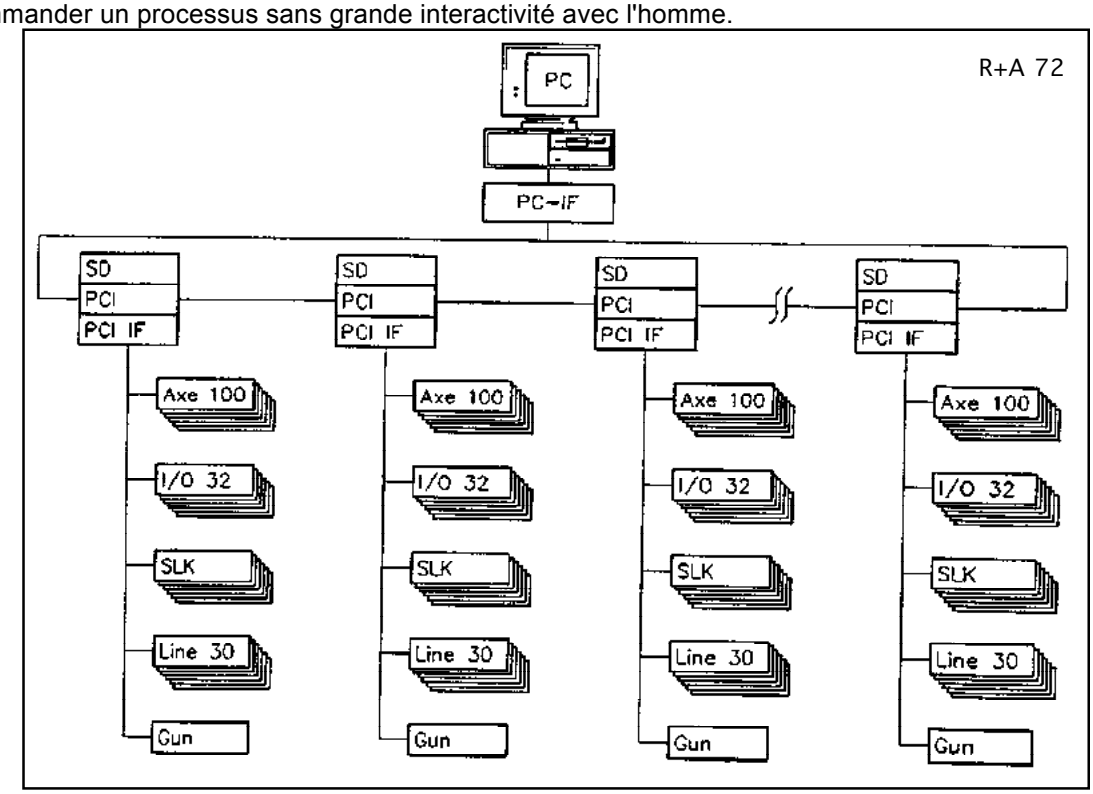


Fig. 3.1.4 Exemple de hiérarchie de commandes dans un îlot de fabrication (doc. Demaurex). On reconnaît dans ce système plusieurs OI du type présenté dans la fig. précédente.

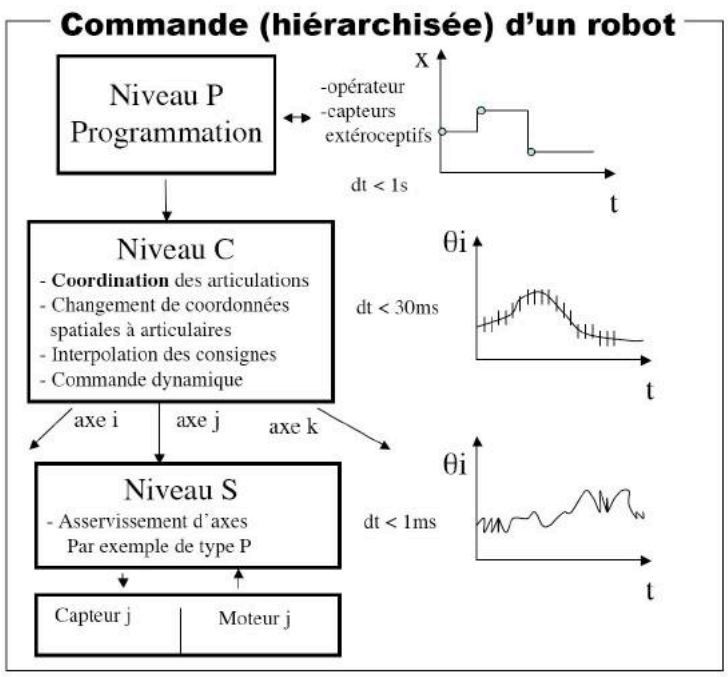


Fig. 3.1.5 Les commandes de R.I. ont généralement trois niveaux

3.1.2 CONTROLE DE ROBOT

Un robot est un système complexe dont le contrôle est assuré par une hiérarchie d'organes de commande. Nous pouvons typiquement reconnaître trois niveaux bien distincts. Le premier, celui de la programmation (P), est très différent fonctionnellement du deuxième, la coordination des articulations (C), même s'il arrive souvent qu'ils se gèrent sur le même matériel. Quant au troisième niveau, les servocommandes (S), il correspond à des fonctions qui sont en général assurées par des équipements propres.

A NIVEAU P

Au niveau le plus abstrait, les actions que le robot doit effectuer sont surtout décrites dans un programme défini durant une phase d'apprentissage. Elles sont aussi influencées par des informations de haut niveau reçues durant l'exécution en provenance soit de l'opérateur, soit, surtout, des capteurs extéroceptifs. Les instructions formant les programmes se succèdent à un intervalle moyen de l'ordre de la seconde pour une tâche typique. Ceci est particulièrement vrai pour les instructions concernant le mouvement.

B NIVEAU C

A un niveau intermédiaire, les articulations du robot sont synchronisées pour garantir des mouvements coordonnés. Des transformations entre référentiels et espaces divers (articulations, atelier, outil, ...) sont assurées , ainsi qu'une interpolation entre les positions commandées par le niveau du programme. Une modélisation dynamique au moins grossière permet de réduire les chocs et les saccades aux endroits critiques, tout en permettant globalement des temps de cycles réduits.

C NIVEAU S

Finalement, on trouve des circuits "classiques" de réglage au niveau de chaque articulation. Les articulations sont réglées en position, en vitesse ou en couple. Ces régulateurs, de type P, PD, PID ou similaires en technologie numérique travaillent en boucle fermée alimentant les actuateurs en fonction des consignes reçues du niveau C et des données des capteurs proprioceptifs.

3.1.3 COMMUNICATION ET RESEAUX DE DONNEES

Qui dit commande hiérarchisée, dit implicitement multiplicité de ressources et nécessité d'interconnexion.

La communication entre composants d'un système de commande hiérarchisé, en milieu industriel, se fait de différentes manières suivant l'application. Chaque fois, il faut satisfaire au mieux les exigences particulières de débit d'information et de proximité des équipements.

On distinguera ici schématiquement 4 classes de moyens de communication (voir fig. 3.1.6).

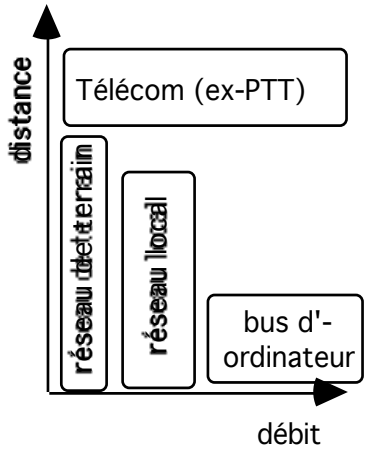


Fig. 3.1.6 Classes de moyens de communication en réseau

Bus d'ordinateur. Les débits d'information les plus élevés sont possibles dans le "coeur" même des équipements, mais ceci restreint fortement la portée possible de la communication. On a typiquement ce mode de travail là ente différents niveaux hiérarchiques d'une commande de robot. Au LaRA, on dispose de ce mode de communication entre processeurs internes de la plupart des robots (ABB, RX, Delta, etc.).

Réseaux de terrain. A l'autre extrême, la communication se fait par débit relativement faible, mais sur de longues distances (jusqu'à 1000 m), au moyen de réseaux de terrain. L'objectif principal est ici d'économiser le nombre de fils physiquement câblés, et plus généralement de limiter les coûts par accès, tout en permettant la communication entre commande et signaux physiques bas-niveau: capteurs individuels, actionneurs tout-ou-rien, ou numériques en boucle ouverte. Au LaRA, on dispose en particulier du CAN-BUS et de ModBuspour ce type d'application. Développé pour l'industrie (en particulier, pour les voitures), le CAN-Bus permet de transmettre, jusqu'à des distances de 1000 m, des messages pouvant comporter jusqu'à 48 bit. La mise à jour se fait en quelques millisecondes. Mais actuellement, ces protocoles tendent à être encapsulé dans des messages Ethernet (par ex. Beckhoff)

Réseaux locaux. Pour des débits plus grands, à moyenne distance, diverses solutions existent. Actuellement, c'est la norme "TCP/IP" qui est la plus répandue. Au LaRA, des communications rapides sont possibles entre équipements à l'échelle d'une place de travail à l'aide de lignes série à

signaux différentiels de norme RS-485 (par ex. Aria, Indel) ou par réseau optique (INFO). Pour les communications à l'intérieur des bâtiments, c'est le réseau Ethernet (10à1000Mb/s) qui s'utilise, sur lequel divers protocoles coexistent.

Réseaux à grande distance (Swisscom etc.). Pour les grandes distances, d'un bâtiment à un autre, ou à l'échelle intercontinentale, les liaisons sont assurées par les services publics (ou privés, depuis la "déréglementation"). En ce qui concerne le débit disponible, une large palette d'offre existe. Les limites ultimes sont posées par la vitesse de la lumière... et par le coût financier du service demandé! Au LaRA, on a accès aux lignes téléphoniques RNIS (128'000 bit/s), ainsi qu'au service informatique du canton de Vaud via lequel l'accès au réseau mondial Internet est en particulier possible à très grande vitesse. Sur celui-ci, il y a notamment le protocole WWW (World-wide web) qui est résident, pour lequel le labo fournissait déjà un serveur d'information dans la première moitié des années nonante (cf. Mosaic, avant Netscape !).

Autres moyens de communication. Il est enfin possible d'établir une liaison entre un équipement et son environnement via les entrées/sorties traditionnelles (par signal booléen, ou par variable scalaire, de type numérique ou analogique). Mais cela sort du cadre de notre discussion car il ne s'agit alors pas typiquement de réseaux (n partenaires possibles) mais plutôt de liaisons point-à-point (1 partenaire par canal). Dans ce contexte, la liaison série de norme RS-232 est encore répandue (1000 m max, débit de 300 à 115kbit/s, typiquement: 9600 bit/s). Pour cette dernière, un logiciel de bas-niveau, à interruptions, a été développé, que l'on a beaucoup utilisé pour l'échange de données entre équipements en contexte Pascal (PC-Rx, PC-Comau, PC-PC, etc.). Actuellement, les outils de base sont intégrés dans Windows.

3.2 PROGRAMMATION DES SYSTEMES DE COMMANDE

Cette section commence par un exemple de tâche d'assemblage. L'exemple illustre le travail de programmation nécessaire pour les API, les OI, et les Robots industriels, qui sont discutés aux points suivants.

3.2.1 EXEMPLE D'APPLICATION: ASSEMBLAGE D'UNE POMPE

Considérons un exemple de montage robotisé: Deux robots coopèrent pour assembler une pompe. Des pièces arrivent, avec une orientation aléatoire sur des bandes transporteuses. Ces dernières sont pilotées par API. Des ordinateurs localisent des pièces et estiment les forces mises en jeu lors de l'assemblage.

Le système assure les fonctions suivantes :

1. Estimation de position et d'orientation des pièces à l'aide d'un système de vision.
2. Prise des pièces sur les bandes transporteuses.
3. Placement de chaque pièce sur un support, assemblage éventuel ou stockage pour utilisation future, suivant ce qui a déjà été assemblé.

A DESCRIPTION DE L'APPLICATION

La séquence qui suit est un segment de l'application. La tâche consiste à prendre un couvercle sur la bande transporteuse, la placer sur le bâti de la pompe, et insérer quatre goujons afin d'aligner les deux pièces. Noter le rôle essentiel joué par l'information venant des capteurs.

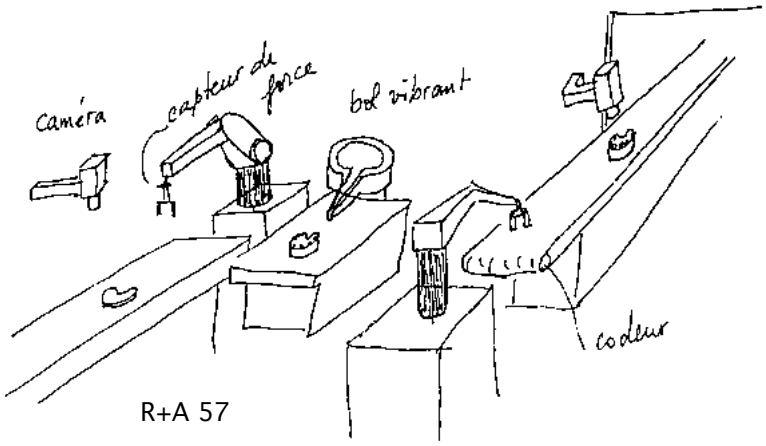


Fig. 3.2.1 Exemple d'opération de montage automatisée

1. Identification, estimation de position et d'orientation par rapport au robot, et inspection d'une pièce (couvercle) qui arrive sur une bande transporteuse. Ceci se fait avec vision par ordinateur.
2. Amener le premier robot, R1, au point de prise prédéfini par rapport au couvercle, dont la position est estimée au point 1. Remarquons que si la bande bouge, il faut continuellement mettre à jour le point de prise en fonction d'un capteur lié à la bande transporteuse.
3. Prendre le couvercle avec une force prédéfinie.

4. Contrôler par l'ouverture de la pince si les dimensions du point de prise sont dans les tolérances ou si l'orientation a été mal estimée. Signaler l'erreur éventuelle.

5. Placer le couvercle sur le bâti en se déplaçant d'abord au-dessus puis s'approchant jusqu'à ce qu'une force de réaction donnée soit atteinte. Tourner la main de manière à annuler les moments résultant d'un mauvais alignement du couvercle. Lâcher le couvercle et mémoriser la position effective pour la suite du travail.

6. En parallèle avec les points précédents, bouger le deuxième robot R2 pour prendre un goujon du vibreur. Amener le goujon en un point au-dessus du premier trou du couvercle, en tenant compte de la position du couvercle mémorisé en 5.

7. Insérer le goujon. Une stratégie pour faire ceci demande de l'incliner légèrement afin d'accroître les chances de trouver le trou. S'il ne tombe pas sur le trou, une recherche en spirale doit s'entreprendre. Lorsque le bout du goujon est dans le trou, celui-ci peut être redressé. Durant ce mouvement, le robot doit appuyer vers le bas avec une force donnée, appuyer de côté pour garder le contact avec le bord et, en même temps, annuler les forces dans la direction perpendiculaire. A la fin de ce mouvement, la position du goujon doit être testée pour garantir qu'elle soit dans les tolérances.

8. Pendant qu'un robot insère le premier goujon, le deuxième robot va en chercher un autre, l'insère à son tour et ainsi de suite jusqu'à la fin. Une synchronisation est nécessaire pour éviter les collisions entre robots.

B TYPES DE CAPTEURS NECESSAIRES

L'exemple de l'assemblage du couvercle d'une pompe fait apparaître la nécessité d'utiliser de nombreux capteurs. Certains sont simples, tels les capteurs de contact, de proximité ou de position, alors que d'autres, tels les capteurs de forces ou ceux d'images requièrent pour l'interprétation de leurs signaux, la puissance de microprocesseurs ou d'OI élaborés.

Capteurs de position

Les capteurs proprioceptifs (p. ex. potentiomètres ou capteurs incrémentaux) permettent de déterminer à chaque instant la position des éléments du robot (articulations et préhenseur) et des bandes transporteuses. Ils sont gérés par la commande des R.I..

Capteurs visuels

Les caméras, typiquement installées au-dessus des bandes transporteuses, permettent de déterminer l'identité et la position des pièces qui passent. Une inspection visuelle par ordinateurs est possible également, assurant ainsi un contrôle de qualité. Une liaison entre OI et RI est alors nécessaire.

Capteurs de contact et de proximité

Des capteurs sur les doigts permettent le contrôle de la force de prise des objets. Ces capteurs sont gérés par une électronique numérique ou par un OI particulier.

Par ailleurs, d'autres capteurs détectent la présence ou l'absence de pièces entre les doigts, l'arrivée d'objets sur le tapis roulant, ou une demande manuelle d'arrêt d'urgence. Ces capteurs délivrent une information binaire, tout ou rien, et sont typiquement gérés par API.

Capteurs de forces dans le poignet

Les erreurs de positionnement du robot, les incertitudes sur la position des objets et les tolérances sur celles-ci, tout conspire pour rendre impossible un positionnement relatif précis, entre organe de préhension et objet à saisir. Ceci est surtout gênant lors d'assemblage à faible jeu. Il est possible cependant d'utiliser les forces générées à mesure que l'assemblage progresse pour faire des mouvements incrémentaux adaptés au résultat final voulu. On parle alors de contrôle de mouvement à accommodation active. Les capteurs de forces y jouent un rôle très important. Ils sont gérés par un OI spécialisé, qui dialogue avec les commandes de R.I..

3.2.2 AUTOMATES PROGRAMMABLES ET ORDINATEURS INDUSTRIELS

Les automates programmables voient leur tâches définies de façon plus simple que les ordinateurs. Le mode de programmation de ces deux types de ressources est discuté dans ce paragraphe.

Il faut cependant bien observer que la distinction entre API et OI paraît parfois excessivement schématique, car l'évolution les rapproche.

Autrefois les ordinateurs de commande de procédés étaient très grands et difficiles à programmer, et l'on avait, à l'opposé, des armoires ou même des salles entières de (logique à) relais. Les API ont rapidement remplacés ces derniers, tout en restant très simples.

Par la suite, l'évolution a permis la réalisation d'ordinateurs de plus en plus simples et petits, alors que les API haut de gamme voyaient leurs capacités augmenter. On est arrivé aujourd'hui au stade où l'API le plus complexe ne se distingue en rien (ni sur le plan matériel, ni sur celui du logiciel) des petits ordinateurs industriels.

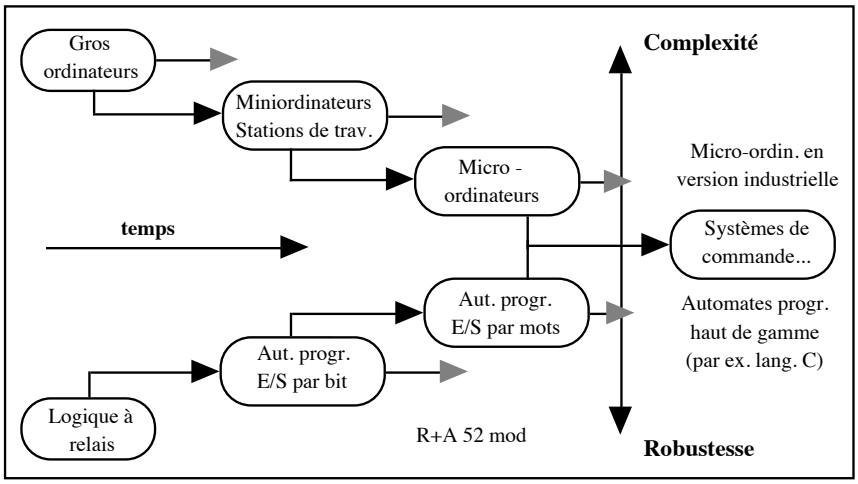


Fig. 3.2.2 L'évolution rapproche les API et les ordinateurs industriels, qui étaient autrefois bien distincts

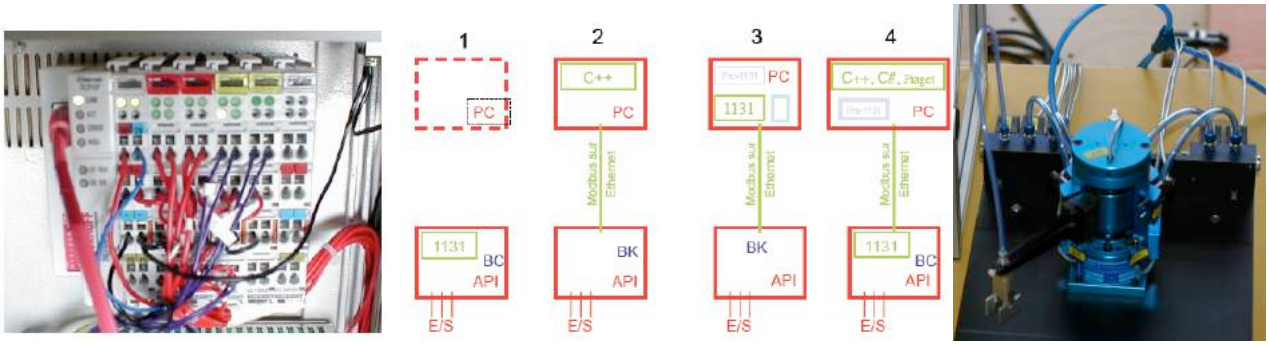


Fig. 3.2.2b API BK9000 du labo (gauche) et architectures possibles pour API. Ceux-ci sont autonomes ou liés plus ou moins directement à un ordinateur. A droite, l'automate mécanique Manutec utilisé pour la manipulation.

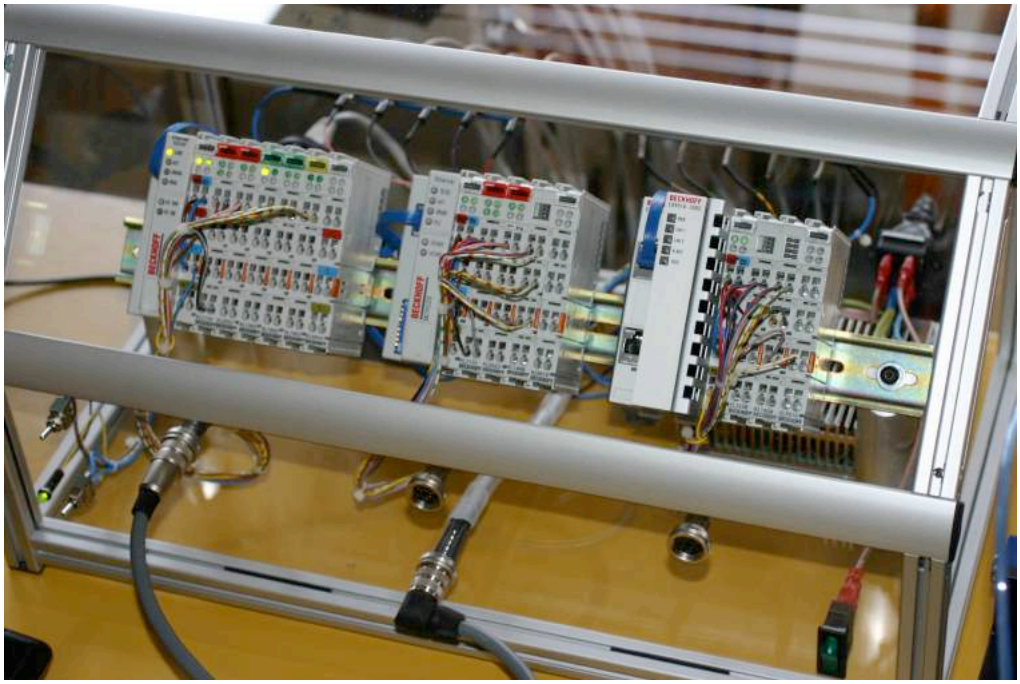


Fig. 3.2.2c API du labo ; BK9000 (gauche, non-autonome), BC9020 (centre, typique) et CX9010-1002 (droite, yc. microcontrôleur).

A MODES DE PROGRAMMATION DES AUTOMATES PROGRAMMABLES

Les API se programment de différentes manières, pour la plupart selon la norme IEC 61'131. Certains modes remontent aux origines de la technique (logique à relais, listes d'instructions, blocs logiques), alors que d'autres résultent de propositions plus récentes, résultant d'une démarche théorique (grafcet) ou de l'influence grandissante de l'automatisation par ordinateur (texte structuré). Pour plus de détails, il existe un polycopié supplémentaire et la manipulation no 25 au laboratoire de robotique et automatisation.

LOGIQUE A RELAIS

Le premier mode de programmation des API recoure à un formalisme traditionnel, de signification intuitivement évidente: les conditions permettant le changement (par exemple passage d'un état "0" à l'état complémentaire "1") de chacune des sorties sont codées par un réseau d'interrupteurs les liant aux entrées. Ces interrupteurs sont eux-mêmes commandés par les signaux d'entrées ou des fonctions internes. C'est de façon graphique qu'un tel mode de programmation permet les meilleurs résultats.

Dans l'exemple de l'assemblage de pompe, ce mode de programmation serait adapté à la mise en route et à l'arrêt du bol vibreur alimentant les R.I. en vis, ou aux enclenchement et déclenchement des bandes transporteuses.

BLOCS LOGIQUES

Si l'on attribue un symbole (un nom) aux signaux externes et internes d'un API, il est commode d'indiquer leur relation par des opérateurs logiques. Au lieu d'une série d'interrupteurs successifs, comme on pouvait l'avoir dans le cas précédent, c'est maintenant une suite d'opérateurs "ET" que l'on va rencontrer. De même, les interrupteurs en parallèle disparaissent au profit de fonctions "OU". Des fonctions nouvelles, telles le "OU exclusif" ou l'opérateur "majorité" peuvent maintenant apparaître. De plus, des blocs à effet mémoire, tels les "flip-flops", sont alors disponibles.

Dans les variantes plus complexes, où des entrées/sorties numériques (directes ou correspondant à des valeurs analogiques) s'utilisent, les blocs opérateurs sont plus complexes, allant par exemple de l'additionneur jusqu'au bloc de régulation PID (proportionnel-intégral-dérivé).

Dans l'exemple de la pompe, les mêmes fonctions, programmées tout à l'heure en logique à relais, pourraient se programmer maintenant en blocs logiques. En plus, on pourrait par exemple asservir les bandes de transports en vitesse.

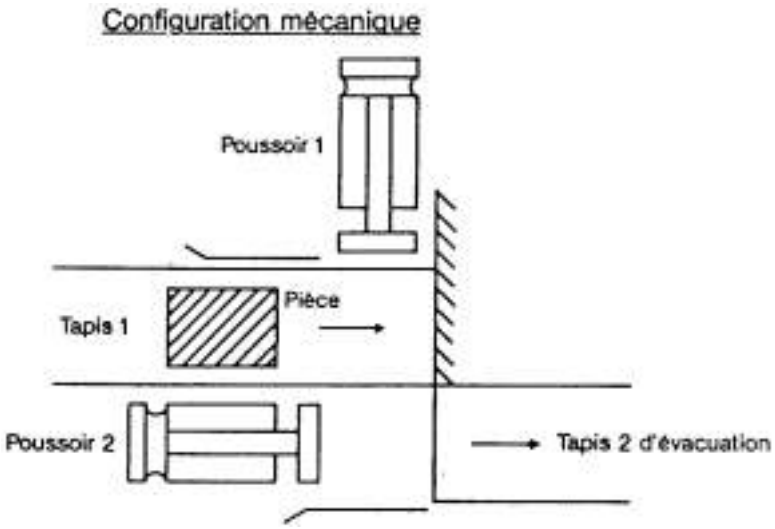


Fig. 3.2.3 Exemple de tâche mécanique à programmer en GRAFCET. La pièce pourrait être le couvercle dans l'application de montage d'une pompe.

GRAFCET

Le GRAFCET et une méthode de définition de tâche proposée par le GREPA (Groupe Équipement de Production Automatisée), un groupe d'experts formé dans le cadre de l'organisme français A.D.E.P.A..

La rigueur de la méthode en ont assuré la diffusion rapide, et ses répercussions sont réelles même aux États-Unis. En particulier, le langage de programmation proposé permet de déclarer de façon explicite les tâches parallèles. Le langage est toutefois limité à des séquences et des combinaisons logiques, aussi il n'est vraiment intéressant que pour les API.

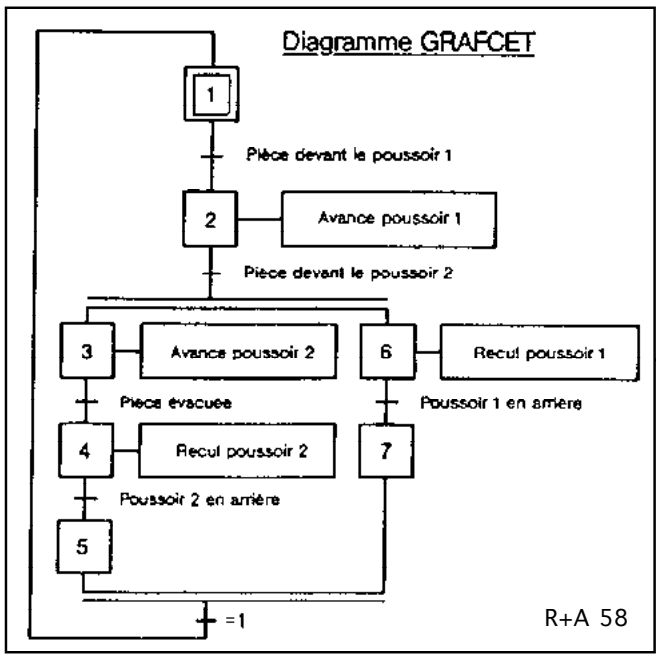


Fig. 3.2.4 Diagramme GRAFCET permettant la programmation

de la tâche définie dans la fig. précédente

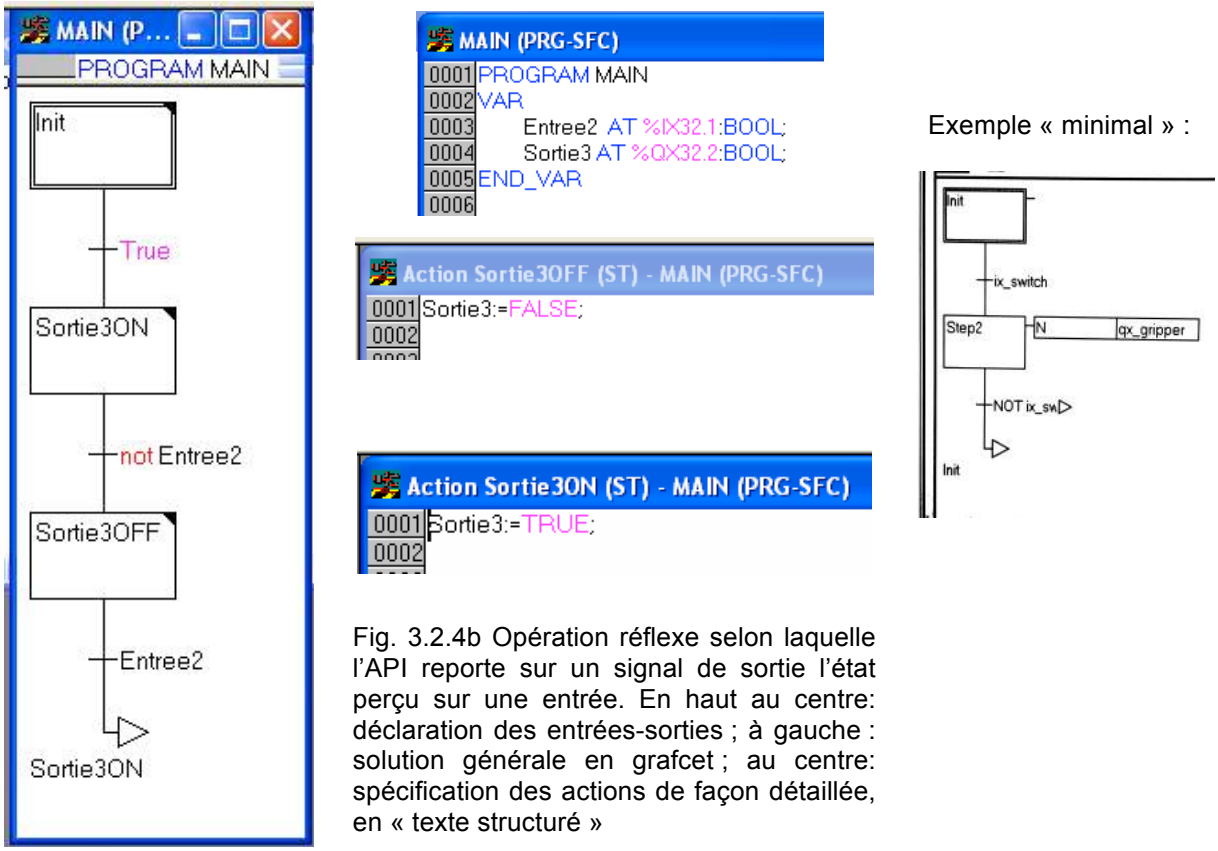


Fig. 3.2.4b Opération réflexe selon laquelle l'API reporte sur un signal de sortie l'état perçu sur une entrée. En haut au centre: déclaration des entrées-sorties ; à gauche : solution générale en grafcet ; au centre: spécification des actions de façon détaillée, en « texte structuré »

Un bloc monostable (« Timer ») existe dans la norme IEC6-1131, qui permet de définir simplement des temps d'attente : ex. MonTimer :TP ; Avec notamment les champs « IN » pour l'entrée, « PT ; PT :**Erreur ! Aucune entrée d'index n'a été trouvée.**=#T1s » pour la durée d'impulsion, et « Q » pour la sortie ; ex. de variable : « MonTimer.Q ».

Du point de vue théorique, un formalisme plus général, qui inclut le GRAFCET, est celui des réseaux de Petri (voir point suivant).

RESEAUX DE PETRI

La théorie des réseaux de Petri est très utilisée pour modéliser les processus industriels parallèles et discrets, et se développe depuis plusieurs décennies dans les groupes de recherches.

Par rapport aux graphes d'états, courants en systèmes séquentiels, la plus grande différence, c'est la présence de "jetons" qui circulent d'une "place" à l'autre, passant les "transitions" si certaines conditions sont satisfaites. Comme on peut le constater sur la fig., une transition se représente par un rectangle, une place par un cercle, et le jeton apparaît comme un point noir.

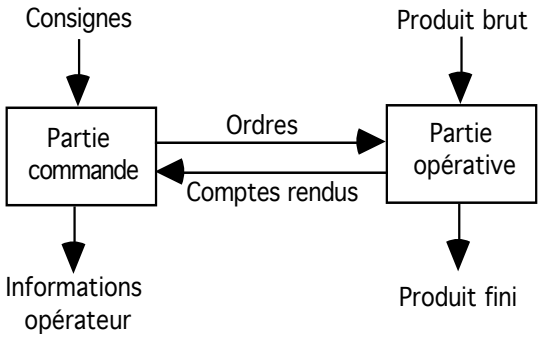


Fig. 3.2.4 ter. Modèle d'un processus complet

De ces conventions de bases, ainsi que de multiples autres considérations dans les cas évolués, on peut tirer un outil très intéressant. Les réseaux de Petri (comme bien d'autres moyens de représentation informatiques) permettent non seulement de décrire une commande, mais aussi de modéliser l'environnement de celle-ci, c'est-à-dire la partie "opératoire" du système.

B PROGRAMMATION DES ORDINATEURS INDUSTRIELS

Les ordinateurs industriels se distinguent quelque peu des ordinateurs généraux, quant à leur mode de programmation, parce qu'ils doivent satisfaire des contraintes particulières. Après avoir mentionné quelques unes d'entre elles, nous verrons quelques exemples de langages utilisés.

EXIGENCES

Les OI doivent satisfaire, outre leur robustesse, à plusieurs exigences particulières.

Interfaçage. L'OI est forcément relié à des entrées-sorties. Nous avons vu dans l'exemple de la pompe divers capteurs destinés à se brancher aux entrées, ainsi que des équipements périrobotiques, tels que les bandes de transport ou le vibreur, susceptibles d'être commandés par les sorties.

Au niveau de la programmation, ces équipements périphériques divers nécessitent souvent la création de pilotes ("drivers") spécifiques.

Temps réel. La contrainte la plus évidente est celle du "temps réel". L'OI est imbriqué dans des procédés de fabrication d'où des données surgissent à tout instant. Par ailleurs, les équipements avals attendent à leur tour des consignes qui ne peuvent être différées. Les programmes doivent pouvoir simplement réagir à des demandes asynchrones d'interruption.

Multitâches. Une place de travail automatisée comprend généralement plusieurs tâches qui se déroulent en parallèles, avec parfois des points de synchronisation. Il est bien sûr très utile pour l'utilisateur de disposer de langages permettant la définition du travail par blocs indépendants, coordonnées par un nombre minimal de structures appropriées (sémaphores, coroutines, etc.).

Communication. L'évolution allant vers une intégration toujours plus poussée des ressources informatiques de l'atelier, il est clair que la fonction de communication gagne en importance. Si pendant longtemps quelques bits parallèles ont suffi, gérés de la même façon que les entrées-sorties mentionnées au §Interfaçage, depuis plusieurs années ce n'est plus le cas. Au minimum, un canal série de type RS-232 est maintenant nécessaire; et souvent, il faut un véritable réseau local, avec ses différents niveaux de protocole.

EXEMPLES DE LANGAGES UTILISES POUR APPLICATIONS INDUSTRIELLES

Aujourd'hui, ce qui est le plus courant c'est probablement l'usage d'un langage général de programmation, complété par des modules écrits en langage assembleur, tournant avec un système d'exploitation spécifique. Les langages les plus utilisés dans ce contexte sont probablement les langages C++, Fortran, Visual Basic ou de type Pascal (Delphi). Un cas qui est en augmentation, c'est l'utilisation de systèmes d'exploitation répartis (cf. Java).

Mais il existe aussi des langages spécialement développés pour le contrôle de procédés. Le plus connu est aujourd'hui Ada, au niveau mondial. En Suisse, le langage Portal a eu un impact et une qualité remarquable, mais est en voie de disparition. Il existe encore toute une série d'autres langages utilisés pour le contrôle de procédés en temps réel, mais il ne sont connus que dans des milieux plus restreints (par exemple, il y a relativement peu de programmeurs qui pratiquent PERL, ou FORTH...).

3.2.3 COMMANDE DES ROBOTS INDUSTRIELS

Les commandes de RI, nous l'avons vu, comprennent plusieurs niveaux. Au niveau le plus élevé, où l'essentiel de la programmation se fait, les tâches à exécuter incluent à la fois des fonctionnalités d'API et d'OI.

Comme un API, la commande de RI doit gérer des entrées-sorties binaires, suivant une logique combinatoire ou séquentielle, impliquant les équipements périrobotiques, et le préhenseur.

Comme un OI, la commande de RI doit permettre la programmation de tâches variées et complexes. Le dialogue avec l'utilisateur et les capteurs extéroceptifs évolués fait appel à des algorithmes et des structures de données diverses, nécessitant que l'on dispose de langages assez riches en instructions.

En plus de ses tâches similaires aux API et aux OI, ce que la commande de robots a en propre, c'est la gestion de positions et de trajectoires dans l'espace. Cette tâche ne peut être transférée à des équipements externes, car elle sous-entend une connaissance détaillée des éléments cinématiques du bras (cf. partie 2 du cours), ainsi qu'une coordination étroite des servocommandes d'articulations.

Pour le montage des couvercles de pompes, l'utilisateur doit par exemple spécifier les mouvements du préhenseur de robot en fonction de positions estimées visuellement, ainsi qu'en tenant compte de l'avance des convoyeurs. On attend de la commande (du langage de programmation) qu'elle (il) gère d'elle-même (de lui-même) le détail des mouvements articulaires.

Voyons d'abord les modes de programmation possibles pour un RI. Ensuite, l'utilité d'un langage approprié sera mise en évidence par un exemple traité de façon détaillée.

A MODES DE PROGRAMMATION DES R.I.

Cinq approches sont schématiquement adoptées pour la programmation des robots. Elles sont adaptées à la définition de tâches de complexité croissante. En général, un mode même puissant peut aussi fonctionner suivant les modes les plus simples.

1 Guidage.

La première méthode consiste à mémoriser essentiellement une suite de valeurs d'articulations ainsi que des signaux de synchronisation pour des équipements externes. Il s'agit d'une programmation par l'exemple ou encore plus simplement par guidage. Cette approche est particulièrement simple et ne nécessite pas vraiment un ordinateur. Elle bénéficie donc déjà d'une longue tradition. Le problème majeur réside dans l'absence de calculs et de sauts conditionnels. Ces "robots" ne s'adaptent donc pas à leur environnement durant l'exécution de leurs programmes.

Les premiers robots programmés par guidage avaient essentiellement comme commande un enregistreur de coordonnées. A l'apprentissage, les coordonnées fournies par l'homme, par exemple à l'aide du déplacement manuel du robot, sont mises en mémoire; ensuite, lors de l'exécution, ce sont les données ressortant de la mémoire qui servent de consignes aux servocommandes individuelles de chaque articulation.

Les robots de peinture sont généralement programmées par guidage. Par guidage également, on peut parfois obtenir le squelette d'un programme, ou l'apprentissage de quelques positions-clefs.

2 Dialogue interactif avec menus.

Un premier pas vers la gestion de tâche complexe se fait lorsqu'un langage de programmation est introduit. Il faut cependant prendre garde à ce que la communication homme-machine reste simple, car l'opérateur utilisant un RI sur la ligne de production est rarement un informaticien.

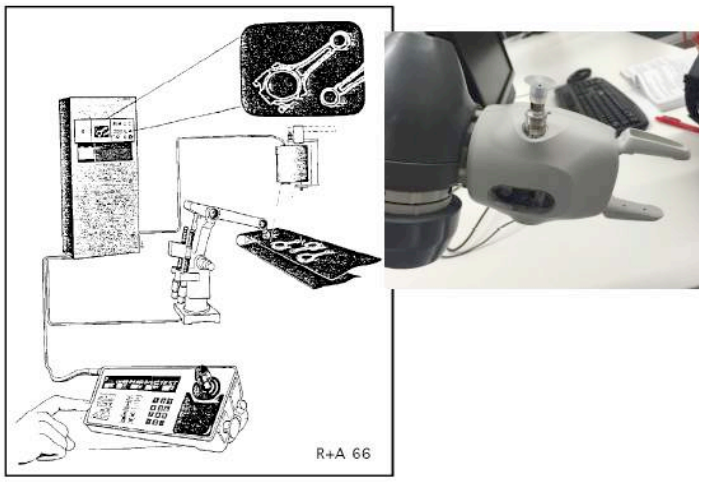


Fig.3.2.5 Le robot est programmé à l'aide de la console portable. Le mode adopté est celui des dialogues interactifs avec menus à 5 entrées, ou celui du guidage. Dans ce dernier cas, c'est la poignée à droite du boîtier qui s'utilise (robot ABB).

La solution passe par l'utilisation de menus interactifs. Les RI les plus répandus, ne sont pas programmés à l'aide d'un terminal classique, avec clavier et écran, mais simplement par une console portable, où un nombre limité de boutons-poussoirs sont disponibles, ainsi que 2 ou trois lignes d'affichage.

La commande est en général suffisamment puissante pour que l'utilisateur dispose de coordonnées multiples (par exemple cartésiennes, articulaires, ou cylindriques). De plus les entrées-sorties (E/S) peuvent s'utiliser pour influer non seulement sur les équipements périphériques mais sur le déroulement du programme lui-même. Des ordres généraux de programmation, tels que la définition de boucles et de sous-programmes, ou tels encore que les accès à des fichiers sont disponibles même sur une simple console portable.

Mais par contre les symboles alphanumériques sont exclus (pas de clavier...), ce qui d'un point de vue pratique restreint l'utilisation du RI à des tâches simples (comment se souvenir par exemple de la signification dans l'atelier de la position 652, stockée dans le programme 356 du secteur 13 de la disquette?).

Si le marché réclame d'abord ce mode de programmation, au niveau de l'utilisateur final, c'est pour deux raisons essentielles. La première, comme on l'a déjà vu, c'est la simplicité de l'interface homme-machine. La deuxième est plus sournoise: une application simple est intrinsèquement plus fiable. D'ailleurs beaucoup d'industriels se méfient des programmeurs, qui promettent souvent la lune (y a-t-il quelque chose qu'un informaticien juge impossible à réaliser?) et ne la livre jamais ("L'application est prête à 95%...").

3 Langages permettant l'utilisation de symboles alphanumériques.

Malgré la réserve concernant la fiabilité des systèmes peu programmables, il faut bien convenir que lorsque les tâches d'automatisation deviennent quelque peu complexes, un terminal complet, et un langage permettant l'utilisation de symboles alphanumériques est un minimum nécessaire.

Les premiers langages développés à ce niveau d'élaboration ont été tirés de langages courants pour OI (BASIC, Pascal, FORTRAN...) avec un nombre variable d'extensions spécialisées.

Les extensions sont tout-à-fait nécessaires pour gérer les servocommandes et les articulations, ainsi que pour réagir aux événements asynchrones survenant dans les équipements périrobotiques.

Afin de permettre un déverminage aisé en conditions réelles, les langages adoptés ici sont généralement interprétés.

4 Langages spécialisés pour R.I..

Mis à part les cas du guidage et des menus interactifs, la programmation de robots repose sur l'emploi de langages explicites. Des instructions sont disponibles pour gérer les capteurs et pour

commander les mouvements du robot. Le robot peut s'adapter dans une très large mesure aux variations même aléatoires de son environnement.

Les langages spécialisés pour R.I. accordent une place prépondérante aux instructions permettant de manipuler les variables de position, ainsi que les trajectoires.

Positions fonction de repères multiples, "cascadés". De nombreuses possibilités existent pour convertir les coordonnées entre référentiels multiples. A l'intérieur de la commande, les positions sont représentées par des matrices de transformation en coordonnées homogènes (cf. partie 2), aussi les cascades de mouvements relatifs, ainsi que les mouvements inverses sont-ils gérés aisément.

Pour bien comprendre comment le problème de la commande de ddl multiples se pose, et surtout comment exprimer un ordre de mouvement, il est utile de revenir un instant sur le cas de la fig. 3.1.1. Dans ce cas, l'axe de mouvement est unique, linéaire, et une forme (syntaxe) possible pour demander le mouvement serait la suivante:

$x = x_0$, ou encore: $x = 10$

où x est une variable correspondant au déplacement de l'élément mécanique entraîné, le signe "=" exprime le désir que l'on a de voir un mouvement se faire, et x_0 (ou 10) est la position désirée. Au début, il se peut bien que le déplacement ne soit pas celui que l'on désire. Par exemple, l'axe est initialement à la position 32. Mais après quelques instants, le système se déplace. C'est le système de commande, en agissant sur la force ou le couple de l'actionneur, qui s'organise de façon à ce que l'égalité soit respectée.

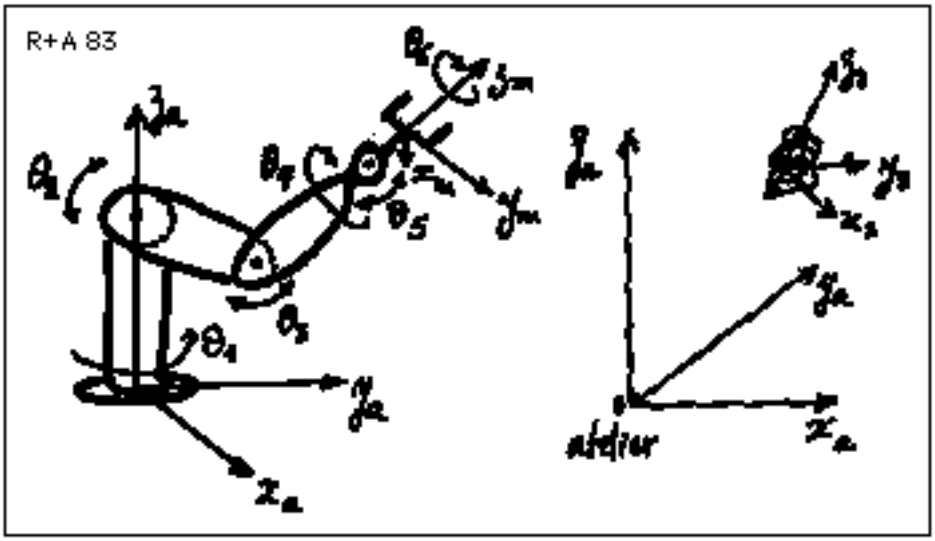


Fig. 3.2.6 L'utilisateur donne la transformation allant de la base du robot au point de prise désiré (par exemple position d'un objet), et le contrôleur gère les variables articulaires

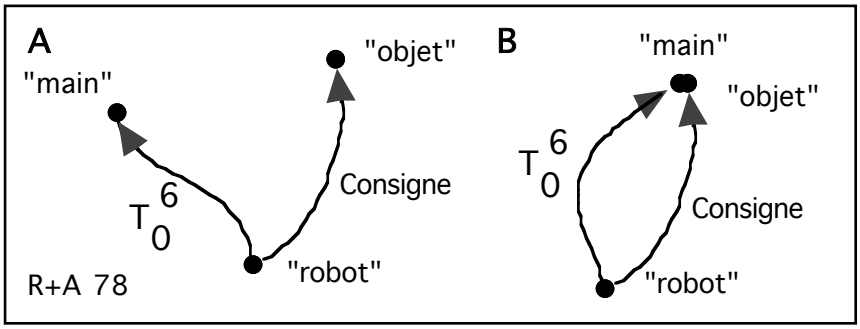


Fig. 3.2.7 Situation avant et après un ordre tel que "$T_0^6 = T_{robot}^{objet}$", ou, dans la syntaxe de V+, "MOVE consigne ».

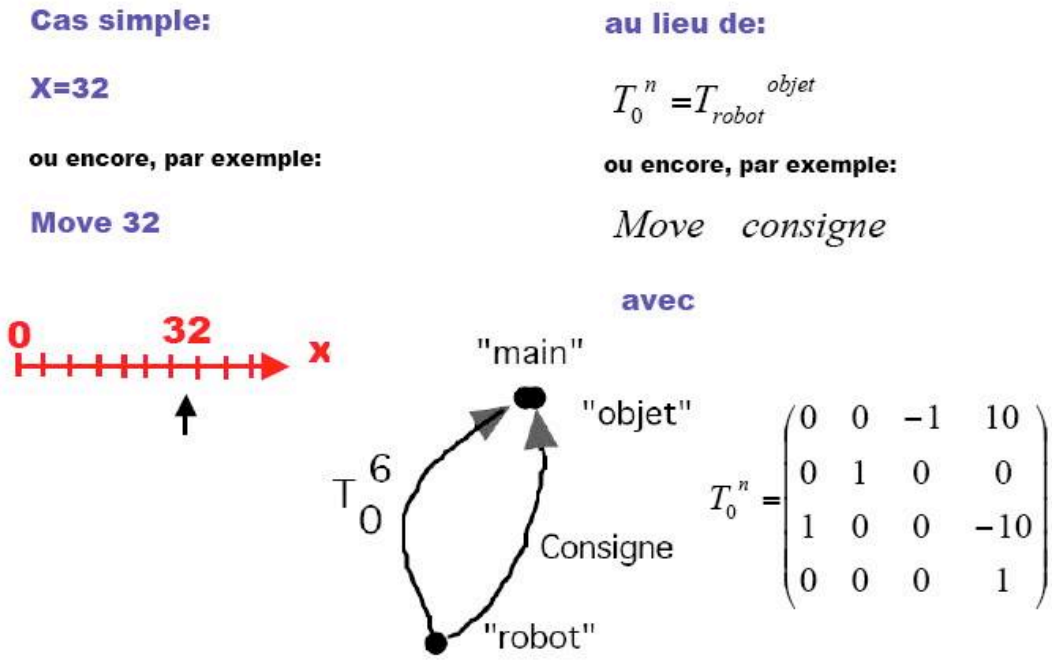


Fig. 3.2.7b Analogie entre la commande, en mode absolu, d'un mouvement pour le cas simple, à 1 ddl, à gauche, et le cas typique pour un robot universel, à droite.

Si l'on a plusieurs ddl, par exemple 6, on peut par analogie proposer une solution concise, ne contenant guère que la consigne donnée sous forme de matrice de transformation, et laisser le soin au système de commande de résoudre l'équation:

$T_0^6 = T_{robot}^{objet}$ ou encore: $T_0^6 = \begin{vmatrix} 0 & 0 & -1 & 10 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & -2.5 \\ 0 & 0 & 0 & 1 \end{vmatrix}$

Avant que le système de commande ne reçoive cette équation, l'extrémité mobile du bras a peu de chance de coïncider avec l'objet. Par contre, dans les instants suivants, le bras se met en marche de façon à ce que l'équation soit vérifiée. La syntaxe présentée ici est élégante. Elle ne correspond pourtant pas de façon immédiate à ce que la plupart des R.I. proposent.

Trajectoire physique et chemin logique. Au début l'utilisateur peut être surpris de la différence qu'il y a entre la trajectoire effectivement parcourue par la main ("chemin physique") et le chemin logique servant à définir la consigne. Les deux chemins aboutissent évidemment sur le même repère. Mais leur point de départ est bien différent.

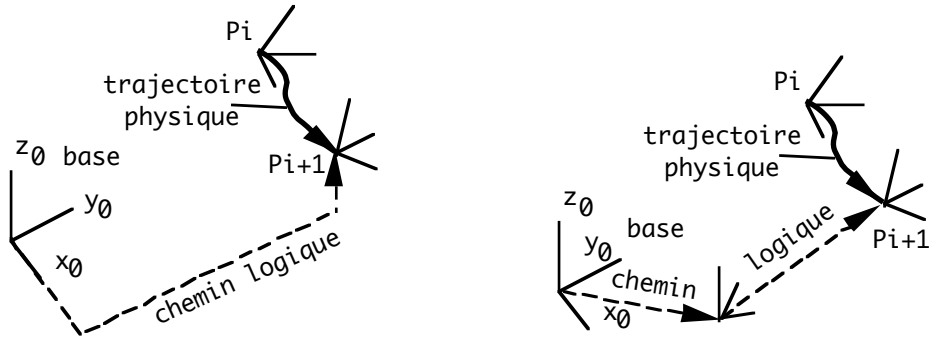
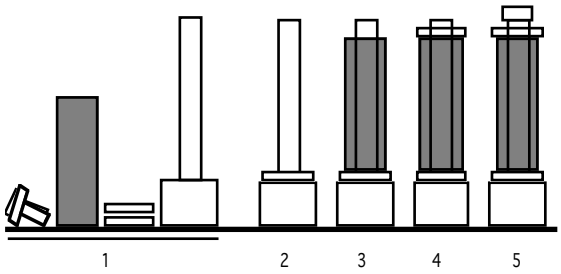


Fig. 3.2.7 c. Chemin logique et trajectoire physique sont différents mais aboutissent au même repère. A gauche la consigne est "simple" (par ex. $T_0{}^6 = T_{robot}{}^{Pi+1}$), alors qu'à droite, elle est "composée" (par ex. $T_0{}^6 = T_{robot}{}^{magasin} \cdot T_{magasin}{}^{Pi+1}$).

Dans le premier cas (trajectoire de l'outil), on part forcément de la position courante (sur la fig. ci-dessus: *Pi*). Dans le deuxième cas (chemin logique), il s'agit de spécifier une consigne, c'est-à-dire une (cascade de) transformation(s), partant obligatoirement de la *base* et qui aboutisse sur le repère voulu (*Pi+1*).

Fig.3.2.7d Définition de tâches par suite d'états



En résumé, on peut dire que la **consigne** est **absolue**, alors que le **déplacement** est **relatif**.

Modification trajectoire en temps réel. Souvent, la possibilité est offerte de modifier en cours d'exécution la trajectoire nominale programmée. Des corrections instantanées sont apportées, en fonction de capteurs fournissant des informations évoluées (données utiles selon plusieurs ddl simultanément). Ceci s'utilise par exemple pour exploiter le long d'une chaîne en mouvement, les tâches décrites dans l'environnement immobile d'un R.I. fixe. Ou encore, cela permet de modifier en fonction de capteurs extéroceptifs une trajectoire définie en valeurs standards.

Modèle dynamique. Certains R.I. ont également maintenant des commandes qui tiennent compte d'un modèle simplifié des éléments dynamiques inhérents à chaque tâche robotisée. Des instructions spécifiques sont alors offertes dans leur langage de programmation.

5 Langages permettant une définition des tâches indépendamment de la cinématique du R.I..

Le plus gros problème, avec les deux modes de programmation précédents, provient du fait qu'il faut être quelque peu expert en programmation et dans les stratégies d'utilisation des capteurs pour pouvoir préparer le robot à une nouvelle application.

Un autre problème provient du fait que les matrices de transformation imposent la définition explicite de toutes les variables spatiales. Souvent la tâche ne présente pas d'exigences si grandes. Par exemple s'il s'agit, dans l'exemple de la pompe, de déposer un couvercle sur une bande de transport, 3 ddl pourrait suffire: l'orientation dans le plan du convoyeur, ainsi que les translations dans ce plan là peuvent rester incertaines, alors que les coordonnées restantes doivent être spécifiées avec précision pour éviter des collisions et des chutes.

De nouvelles stratégies se définissent pour dispenser l'utilisateur de robot d'être un spécialiste en programmation. Ainsi on recherche maintenant un mode de programmation qui traite directement de la tâche à faire, plutôt que des mouvements spécifiques à chaque robot pour exécuter cette tâche. Le problème réside ici dans la difficulté de modéliser l'environnement du robot. Il ne suffit plus d'adjoindre des systèmes de coordonnées aux objets, mais il faut de plus en définir complètement les volumes, afin de calculer automatiquement toutes sortes de grandeurs: trajectoires sans collisions, points de prises, etc..

Ce mode de programmation n'est encore pas disponible en pratique, mais se limite à être un objectif pour de futures solutions.

6 Conclusion

Les approches basées sur le guidage et la programmation par des langages pour robot sont maintenant largement adoptées pour les robots industriels commercialement disponibles. Par ailleurs, certains fabricants (p. ex. IBM, ou, dans une certaine mesure, Automélec et Demaurex) tentent de satisfaire aussi bien le programmeur chevronné, en lui offrant un répertoire très riche d'instructions élémentaires, que l'opérateur néophyte en programmation, pour lequel des commandes évoluées sont prédéfinies à l'aide du même langage, et mises à disposition par menus interactifs.

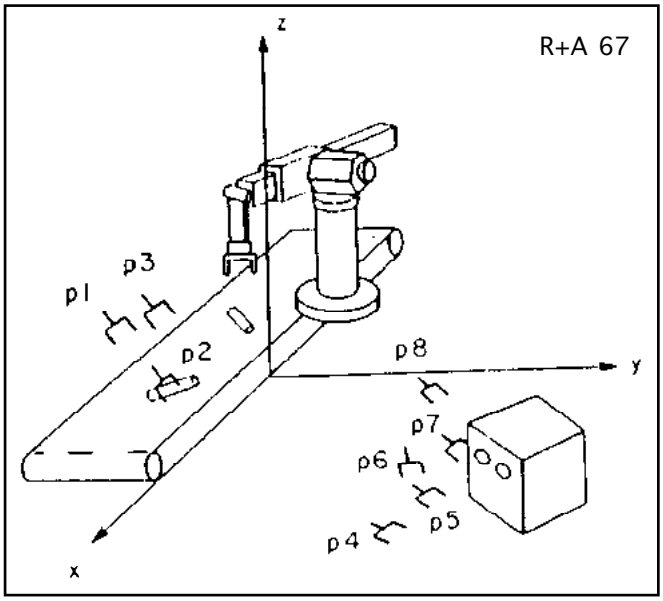


Fig. 3.2.8 Montage "simpliste" de 2 cylindres. Solution par guidage.

Les divers modes de programmation peuvent s'observer sous un angle différent de celui que nous avons adopté jusqu'ici. En particulier, J. Latombe propose une différence entre approches définissant la tâche à des niveaux de complexité croissante: niveaux actionneur, préhenseur, objet, objectif...

B EXEMPLE DE PROGRAMMATION DE R.I. PAR LANGAGE SPECIALISE: PRISE DE CYLINDRES

Comme exemple d'application, considérons l'insertion de deux cylindres dans un cube (voir fig.3.2.4). Cette opération est représentative de beaucoup de tâches d'assemblage (nous avions jusqu'ici l'assemblage d'une pompe. Il est un peu malheureux de changer d'exemple; mais le lecteur aura peut-être la curiosité d'imaginer que les vis du premier exemple se transforment maintenant en cylindres, que les cylindres soient localisés visuellement comme tout-à-l'heure les couvercles, et qu'enfin le bâtis de la pompe se métamorphose en cube...).

PREMIERE SOLUTION

```
MOVE    p1      ; la pince s'approche du cylindre,
MOVE    p2      ; se déplace en position de prise,
CLOSEI          ; se ferme,
MOVE    p3      ; lève le cylindre verticalement,
MOVE    p4      ; approche un trou, légèrement de biais,
MOVE    p5      ; avance jusqu'au contact,
MOVE    p6      ; redresse le cylindre,
MOVE    p7      ; l'insère,
OPENI           ; relâche la pièce,
MOVE    p8      ; s'éloigne.
```

Un tel programme pourrait être exécuté par la plupart des robots industriels, et il indique aussi clairement quelles en sont les limites. Rien n'est prévu pour les tolérances (erreurs de position et forces mises en jeu). Rien n'est prévu pour stocker la position effective des objets. Lorsque l'insertion dans le premier trou est faite, l'information de position devrait être conservée pour faciliter l'assemblage du deuxième cylindre. Si le robot est déplacé, il faut tout réapprendre. De même pour le deuxième cylindre, toutes les positions doivent être à nouveau définies, avec des valeurs légèrement différentes. Ce qui manque, c'est la structure de la tâche.

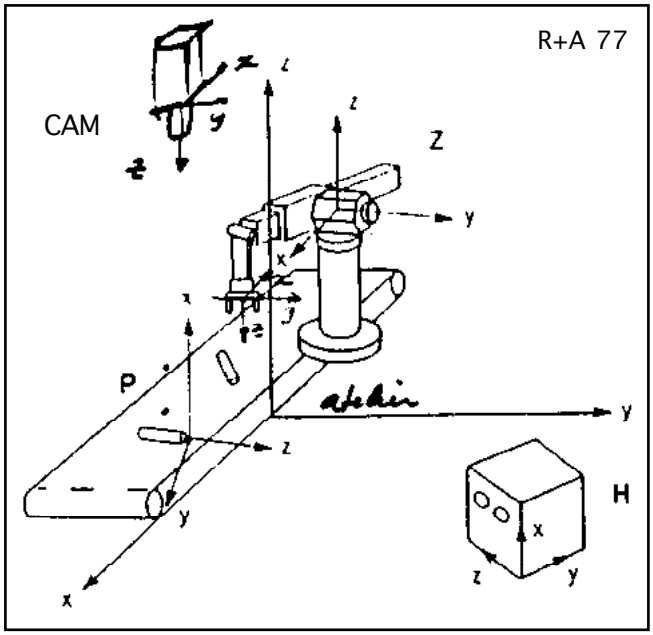


Fig. 3.2.9 Assemblage de 2 cylindres. De nombreux repères ont été définis.

SOLUTION POSSIBLE SUR DES ROBOTS GERANT DES REFERENTIELS

Certains langages pour robots permettent facilement d'exprimer la position et l'orientation d'objets quelconques dans l'espace, tant par rapport à la base standard du robot, que par rapport à des repères librement définis.

Bien qu'ils requièrent un certain apprentissage, ces langages se révèlent rapidement d'emploi facile, et ils sont seuls à permettre de programmer des tâches d'assemblage de complexité réaliste. Comme précédemment, la tâche est programmée ici en V+. Mais cette fois, on tire parti des instructions qui permettent de structurer la tâche. Les mots-clefs du langage sont en majuscules, alors que les symboles introduits par l'utilisateur sont en minuscules. Une autre version, écrite en Val3 est aussi disponible au laboratoire. Notons dans ce dernier cas que la pratique a rattrapé la théorie : le fournisseur fait un usage abondant des graphes de transformation. Dans ce cas, pour représenter les repères, les « points » accompagnés d'un nom, que nous avions dans la partie 2 du cours, sont remplacés par des rectangles dans lesquels le nom du repère est inscrit. Pour le reste, notamment pour les flèches décrivant les transformations, ou pour les règles d'équivalence, tout est identique.

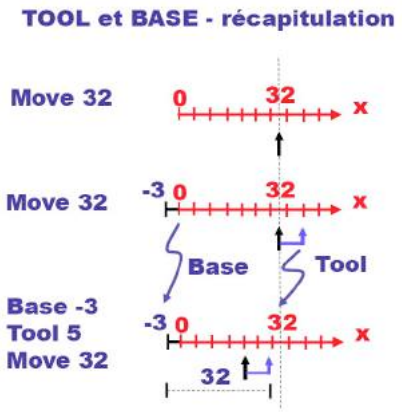


Fig. 3.2.9b Illustration du principe des ordres Base et Tool avec un « robot » à 1 ddl

Programme "CYLINDRES", (écrit en langage de type Val+)

```
1   TOOL      e                ; définition de (la forme de) la pince utilisée. TOOL redéfinit
                                 l'extrémité mobile du bras
2   FOR  i = 1 TO 2  BEGIN     ;
3   ; début d'un cycle qui sera exécuté deux fois
4   CALL      appellecaméra    ; dialogue afin de connaître PC (position du cylindre vu,
                                 par rapport à la caméra)
5   SET       p = cam : pc     ; définit la position du cylindre par rapport à l'atelier (voir
                                 graphiques)
6   BASE      INVERSE(z):p     ; BASE permet de redéfinir ce qui constitue l'extrémité fixe
                                 du bras; celle-ci est maintenant définie sur le cylindre
7   MOVE      pa               ; la pince se déplace en position d'approche,
8   MOVES pg                   ; puis de prise du cylindre (en ligne droite),
```

```
 9   CLOSEI                    ; ferme la pince, avec effet immédiat
10   TOOL    e : INVERSE(pg)   ; définit l'extrémité mobile du bras sur le cylindre (voir fig.
                                 3.2.12)
11   MOVE    pd                ; déplacement en position de départ
12   SET     ht=hr[i]          ; HT est la position du trou courant par rapport au cube (
                                 le cube a 2 trous)
13   BASE    INVERSE(z): h : ht ; H est la position du cube par rapport à l'atelier et
                                 l'instruction redéfinit l'extrémité fixe du bras  sur le trou
                                 courant
14   MOVE    pha               ; le cylindre se déplace en approche du trou courant,
15   MOVE    pch               ; puis jusqu'au contact,
16   MOVE    pal               ; s'aligne sur l'axe du trou,
17   MOVES pn; et s'insère (en ligne droite);
18   OPENI                     ; ouverture immédiate de la pince
19   BASE  INVERSE(z): h : ht : pn ; l'extrémité fixe est redéfinie sur le cylindre
20   TOOL    e                 ; l'extrémité mobile est redéfinie au bout de la pince
21   MOVE    pa                ; la position prédéfinie pour l'approche (ligne 7) s'utilise ici
                                 pour quitter le cylindre sans collision.
22   END
```

Même début du programme « Cylindres », cette fois écrit en Val-3 :

```
trTOOL = trE                  // 1 définition de (la forme de) la pince utilisée.
                              // 1bis TOOL redéfinit l'extrémité mobile du bras
for  i = 1 to 2               // 2
                              // 3 début d'un cycle qui sera exécuté deux fois
call appelleCamera()          //4 dialogue pour connaître trPC (position
                              // 4bis     du cylindre vu, par rapport à la caméra)
trP = trCAM*trPC              // 5 définit la position du cylindre par
                              // 5bis   rapport à l'atelier (voir graphiques)

trBASE=!trZ*trP               //6 BASE permet de redéfinir ce qui
                              // 6bis   constitue l'extrémité fixe du bras; celle-ci est
                              // 6ter    maintenant définie sur le cylindre
 myMovel(trPA)                // 7  la pince se déplace en position d'approche,

 myMovel(trPG)                // 8  puis de prise du cylindre (en ligne droite),
```

```
close()                    // 9 ferme la pince, avec effet immédiat
trTOOL= trE *!trPG         // 10 définit l'extrémité mobile du bras
                           // sur le cylindre (voir fig. 3.2.12)
myMovel (trPD)             // 11 déplacement en position de départ
```

…

avec les déclarations préalables suivantes :

```
// Déclarations pour le programme "Cylindres », en Val-3
num i                          // compteur pour boucle for - numéro de cycle
trsf trP, trCAM, trZ, trTOOL, trBASE, trE…
program myMovel(trsf trT)  // utilise trBASE, trTOOL
                           // configuration libre et mNomDesc
begin
 tool myTool ,             // outil
 point mypBase, mypNul
 frame myfBase
 myfBase=world
 myfBase.trsf=trBASE
 mypNull={{0, 0, 0, 0, 0, 0}, {sfree, efree, wfree}}
 mypBase=compose(mypNul,world, trBASE)
 myTool=flange
 myTool.trsf=trTOOL
 movel(compose(mypBase,myfBase,trT),myTool,mNomDesc)
end
```

REMARQUES

Utilisation des instructions BASE et TOOL. Les instructions BASE et TOOL permettent de choisir les repères correspondant aux extrémités fixes et mobiles du bras. Elles peuvent s'utiliser de façon plus ou moins "subtile".

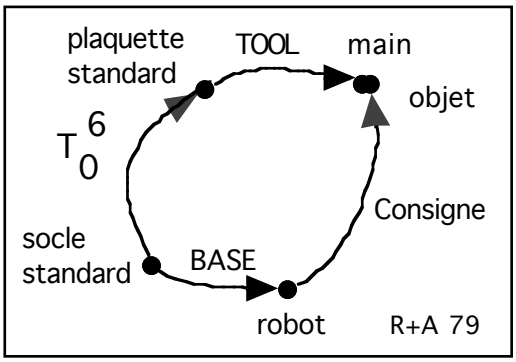


Fig. 3.2.10 Les instructions BASE et TOOL servent à décrire les prolongements "naturels" du bras, tels que le socle auquel le robot est fixé, ou l'outil que le robot manipule.

En principe ces instructions s'utilisent pour tenir compte de légères variations relatives au montage du robot sur son socle, ainsi que de la forme particulière de différents préhenseurs.

Mais TOOL et BASE permettent en plus de gérer simplement, dans la chaîne de transformation qui constitue habituellement la consigne, tous les repères momentanément solidaires des extrémités du robot. Du côté fixe, on pourra par exemple, avec BASE, englober un magasin, où la pièce manipulée doit être déposée. Du côté mobile, on pourra, en plus de la description du préhenseur, rajouter peut-être le repère décrivant la pièce manipulée.

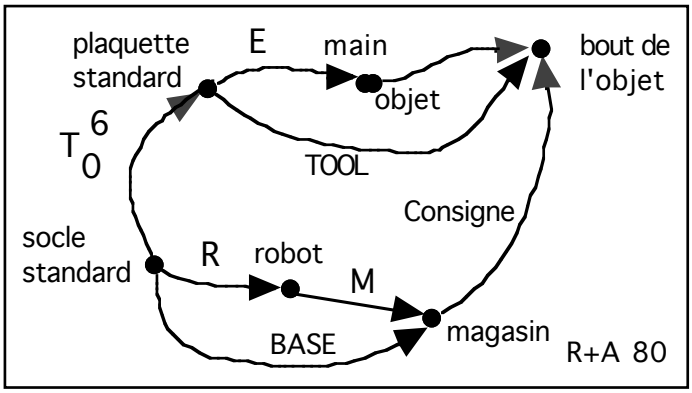


Fig. 3.2.11 Les instructions BASE et TOOL permettent de redéfinir des extrémités "généralisées" pour la chaîne cinématique que représente le bras. Celui-ci se prolonge alors à chaque bout d'un nombre parfois grand de transformations connues.

Ceci conduit parfois à des situations étonnantes de simplicité, pour l'expression de la position de l'extrémité mobile par rapport à l'extrémité fixe. Ainsi à la ligne 10 du programme "Cylindres", cette position se décrit par la matrice identité!

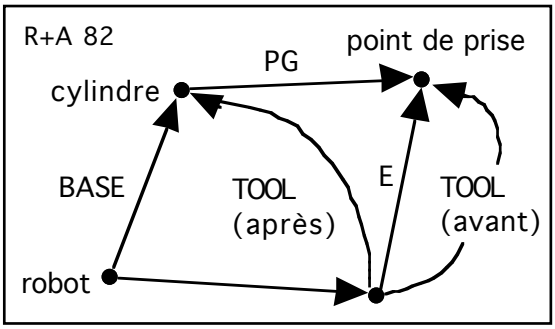


Fig. 3.2.12 Après l'instruction de la ligne 10 du programme "Cylindres", les extrémités généralisées du bras, l'une fixe et l'autre mobile, sont confondues.

Variables prédéfinies. Ce programme contient beaucoup de variables prédéfinies. CAM et Z sont définis lors de changement de configuration de la caméra ou du robot par rapport à l'atelier. E décrit un préhenseur une fois pour toutes. Les autres positions (PA, PG, etc.) peuvent se définir interactivement, par essai avec le robot, ou explicitement, tant leur contenu est simple (logique). En observant le dessin de la station, on constate par exemple que PC est égal à PA translaté d'une distance d selon X. La valeur de d ne peut pas être donnée numériquement ici, car le dessin n'a pas d'échelle, mais on peut dire qu'il sera négatif, et sa grandeur devrait être de l'ordre du diamètre des cylindres.

Exemple de définition d'une position. Voyons plus en détail, comme exemple, ce que vaut PA. PA doit contenir la consigne qui amène le préhenseur au dessus du cylindre, avec une orientation qui prépare la prise. La position doit être suffisamment haute pour qu'on l'atteigne sans collision en venant d'un peu n'importe où.

La consigne doit toujours être donnée entre extrémité fixe et extrémité mobile. Or à la ligne 7 du programme, où cette (matrice) variable s'utilise, l'extrémité fixe du bras est précisément définie sur le cylindre! Et l'extrémité mobile sur le préhenseur! Notre tâche est donc bien simple.

Commençons par la position. Pour fixer l'échelle, et afin d'être concret, admettons que le cylindre ait une longueur de 5 cm. Pour PA, $OXYZ_{préhenseur}$ peut être positionné en 10 cm selon $X_{cylindre}$, 0 cm selon $Y_{cylindre}$ et -2.5 cm (milieu du cylindre) selon $Z_{cylindre}$. Quant à l'orientation, elle peut être décrite également en observant la station de travail. $X_{préhenseur}$ doit être aligné avec $Z_{cylindre}$.

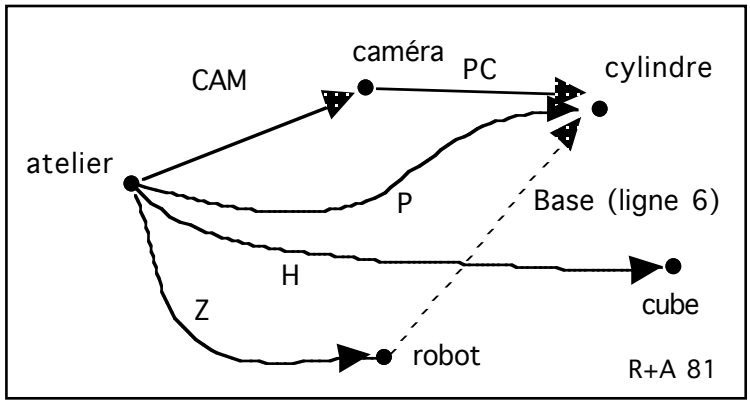


Fig. 3.2.13 Analogie graphique pour les transformations intervenant dans les lignes précédant l'utilisation de la position PA, dans le programme "Cylindres".

Choisissons arbitrairement qu'ils sont dans le même sens. Il en résulte que $Y_{préhenseur}$ est aligné avec $Y_{cylindre}$. Enfin, $Z_{préhenseur}$ est aligné avec $X_{cylindre}$, mais en sens inverse. Sous forme matricielle, nous avons donc:

$$PA = \begin{pmatrix} 0 & 0 & -1 & 10 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & -10 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Si l'on ne veut pas définir PA ainsi, analytiquement, et que l'on dispose du robot, d'autres solutions, interactives, sont alors possibles. Par exemple après les définitions de TOOL et de BASE faites selon le programme aux points 1 et 6, on amène le préhenseur en position d'approche, à l'aide de la console portable et de ses boutons-poussoirs. Lorsque la position est satisfaisante, à l'oeil, on peut alors taper sur le terminal:

```
DO  SET  pa = HERE    ; "DO" est un préfixe qui permet de transformer une instruction de
                        programme en commande à effet immédiat. HERE est une fonction du système
                        qui renvoie la position courante de l'extrémité mobile par rapport à l'extrémité
                        fixe.
```

Dialogue avec l'équipement de vision. A la ligne 4, le programme "Cylindres" fait appel à un autre programme (sous-programme "appelle.caméra"). Dans la configuration actuelle du laboratoire, "appelle.caméra" envoie typiquement une chaîne de caractères ("POSITION?") sur une ligne série à un PC-AT. Celui-ci, qui est équipé d'un numériseur d'image, analyse l'image de la caméra, et répond au message reçu ("POSITION?") par l'envoi des coordonnées, en pixels, du cylindre dans son champ de vue. Le sous-programme fait la conversion de pixels en mm, et utilise cette information pour mettre à jour la position PC. Cette variable est globale, aussi le programme "Cylindres" pourra-t-il l'utiliser au retour d'"appellecaméra".

Nom des variables. Les variables utilisées ici ont des noms très courts (Z, P, CAM, etc.). Cela est admissible parce que le programme s'accompagne de plusieurs figures et graphiques où la signification de ces symboles apparaît de façon évidente. Si ces auxiliaires ne sont pas prévus, il vaut mieux utiliser des noms plus explicites. Par exemple, dans le programme "Cylindres", on aurait pu utiliser le symbole "position.du.robot" au lieu de "z". Ou mieux encore, on aurait pu choisir "t.atelier.robot", ce qui est complet et rappelle la syntaxe utilisée dans la partie 2 du cours: $T_{atelier}^{robot}$. Les matrices de transformation portaient le nom des repères qu'elles reliaient; non seulement le repère de destination y apparaissait, mais aussi celui de départ (autrement dit, il faut spécifier non seulement la position qui nous intéresse, mais encore par rapport à quel référentiel on la décrit).

3.3 COORDINATION DES ARTICULATIONS D'UN ROBOT

3.3.1 VUE D'ENSEMBLE

L'étage de coordination transmet à chaque servocommande des consignes successives aussi voisines que possibles, afin que les écarts entre position effective et position demandée soit en tout temps négligeables. L'étage de coordination génère les consignes qu'il transmet au niveau S par interpolation des consignes reçues du niveau P.

Divers types d'interpolation sont adoptés de cas en cas. En général, l'interpolation est linéaire dans l'espace des coordonnées articulaires. Par rapport au temps, l'interpolation varie suivant la loi de déplacement utilisée. Les plus courantes sont celles dites "à profil trapézoïdal" ou à "profil tabulé".

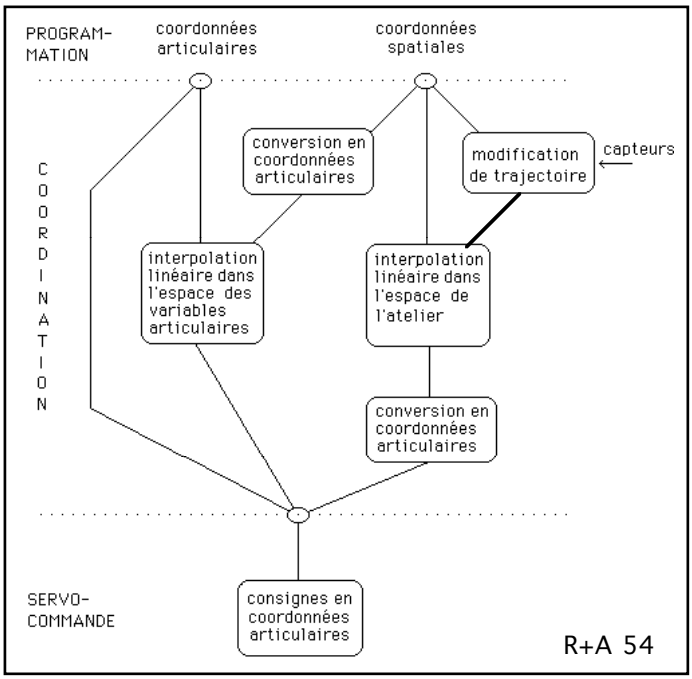


Fig. 3.3.1 Vue d'ensemble de l'étage de coordination

Pour la plupart des robots récents, le contrôleur peut assurer un changement de coordonnées en temps réel. Typiquement, il s'agit là de résoudre à haute vitesse le "problème cinématique inverse" du bras (cf. partie. 2). L'interpolation peut alors se faire dans l'espace des coordonnées de l'atelier. En général, cette interpolation est linéaire. Si un mouvement autre que linéaire est désiré, il faut, au niveau P, calculer des points rapprochés. On parle alors de mouvement procédural. Ceci revient à faire l'approximation de la trajectoire voulue (p. ex. circulaire) par une suite de petits segments de droite.

On distingue le mode d'interpolation "point-à-point", d'un autre mode d'interpolation, dit de "trajectoire continue". Dans le premier cas, le bras s'immobilise brièvement à chaque position demandée par le niveau P. Dans le second, le passage d'un segment de trajectoire au suivant se fait sans arrêt, sans choc, ni saccade.

L'étage de coordination d'un contrôleur de robot travaille généralement en boucle ouverte. Mais parfois aussi, un canal d'entrée permet de modifier la trajectoire courante en fonction de capteurs extéroceptifs.

Certains robots industriels n'ont pas d'étage de coordination dans leur unité de contrôle. Il appartient alors à l'utilisateur de gérer explicitement et dans le détail la synchronisation des articulations en agissant au niveau P. Les possibilités de coordination sont alors limitées.

Il existe maintenant des commandes tenant compte en temps réel de paramètres dynamiques. L'étage de coordination modifie alors les paramètres limites des servocommandes en fonction des inerties instantanées (du bras, compte tenu de sa configuration, et de la charge, compte tenu de déclarations faites par l'utilisateur ou de certains capteurs, au niveau P).

3.3.2 Lois de mouvement pour articulations individuelles. Interpolation par rapport au temps.

Le programme envoie des consignes qui sont relativement distantes, tant du point de vue spatial que temporel. Il faut les interpoler afin que la trajectoire, c'est-à-dire la réponse combinée de toutes les articulations soit prévisible. Ce paragraphe se préoccupe de l'interpolation par rapport au temps de chaque articulation prise isolément.

Nous verrons tout d'abord le problème de l'interpolation dans son principe. Puis nous examinerons des lois de mouvements calculées en temps réel, ainsi qu'un autre type de solution, où un profil type, normalisé, est résident en mémoire.

A Principe

Les lois présentées ci-dessous ne sont pas directement utilisées au niveau "asservissement", mais servent au niveau "coordination" à modéliser le comportement des régulateurs individuels de chaque articulation. Elles sont relativement peu précises (par exemple les déformations des éléments sont en grande partie négligées), mais elles suffisent à permettre à l'étage C de coordonner valablement les articulations. De légères erreurs de modélisation à ce niveau de commande sont parfaitement tolérables, car les servocommandes qui se trouvent en aval travaillent en boucle fermée, et peuvent compenser non seulement certaines erreurs du modèle, mais aussi l'influence de nombreuses perturbations externes.

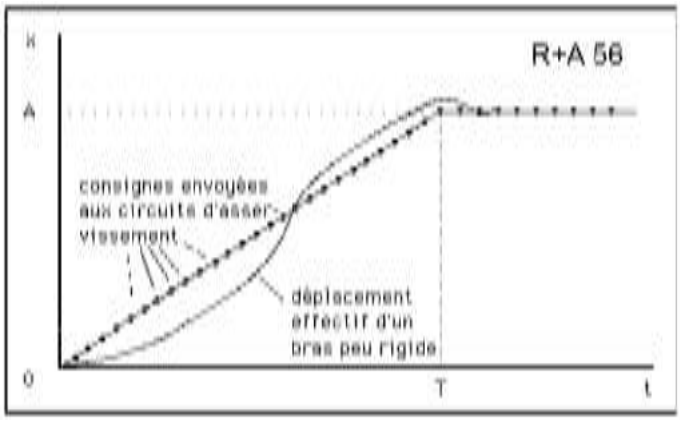


Fig. 3.3.2 Les lois de déplacement servent à coordonner les articulations. Elles ne tiennent pas compte des mouvements effectifs du bras, comprenant en particulier des déformations élastiques.

On peut schématiquement décomposer les différents modes d'interpolation en trois éléments: l'amplitude, la forme et la durée.

L'amplitude du mouvement à réaliser, A, est fixé par le niveau P. Ce paramètre est bien respecté, même si certains modes (en particulier, l'interpolation dite "à trajectoire continue", que nous aborderons par la suite) y font parfois une légère entorse.

La forme peut varier dans une large mesure. Ce point est développé dans la suite. Les variantes sont nombreuses mais elles ont toutes pour objectif d'offrir un bon compromis entre la minimisation du temps de cycle, la minimisation des oscillations et dépassements, ainsi qu'une simplicité maximale des calculs.

La durée du mouvement d'interpolation, T, permet de faire varier, pour une amplitude et une forme donnée, l'erreur de dépassement à la fin du mouvement. En même temps qu'une forme est définie, une équation est établie afin de calculer la durée. Les paramètres de cette équation peuvent être multiples, prenant par exemple en compte la fréquence propre d'oscillation de l'élément entraîné, sa vitesse maximale, ou l'accélération la plus grande que l'actionneur peut provoquer.

Demaurex a par exemple proposé une équation qui met en relation ces divers éléments:

$$D < \frac{1}{\omega_0^m} \sum_{1}^{n} \sigma_i$$

où D est l'amplitude du dépassement, ω_0 est la pulsation propre de l'élément mécanique; m est l'ordre de la dérivée du déplacement où les premières discontinuités apparaissent; σ_i sont les amplitudes (en valeur absolue) des discontinuités y apparaissant, et n leur nombre.

Les termes σ_i sont représentatifs tout à la fois de la forme de la loi d'interpolation utilisée, et de la durée T du mouvement. Ainsi par exemple pour la loi à vitesse constante ci-dessous, nous avons deux discontinuités σ_1 et σ_2 qui toutes deux valent A/T.

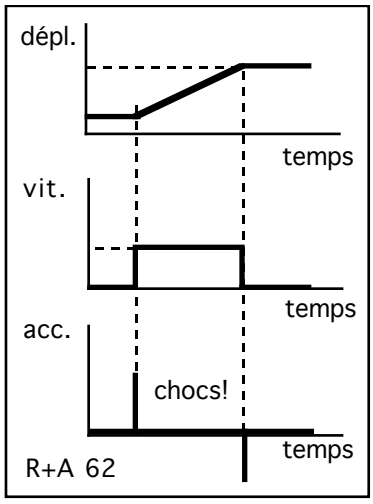


Fig. 3.3.3 Loi de mouvement à vitesse constante.

B MOUVEMENTS CALCULES

Le mouvement calculé le plus courant prend, par rapport au temps, la forme d'un profil trapézoïdal. Il résulte d'une combinaison de mouvement à vitesse ou à accélération constante.

VITESSE CONSTANTE

La loi la plus simple est celle du profil de vitesse constant.
Cette loi est assez schématique, et quelque peu irréaliste, d'un point de vue strict, par le fait que l'accélération présente des pointes d'amplitude infinies. Cependant, elle est pratique, d'une part à

cause de la simplicité des calculs qu'elle entraîne, et parce que, pour la plus grande partie des mouvements, elle décrit assez bien la réponse d'une articulation où les forces de frottement dominent les termes inertiels.

ACCELERATION CONSTANTE

La loi de mouvement à accélération constante décrit assez bien les phases d'accélération de et de décélération des mouvements. Dans ces zones, on peut travailler à couple ou force constante, qui sont, dans le plus rapide des cas, liées aux forces et couples limites de l'actionneur.

La loi précédente était constante au niveau de la première dérivée du déplacement, c'est-à-dire de la vitesse. Cela donnait des chocs. La loi observée maintenant est plus douce, n'entraînant que des saccades, car elle est constante (par morceaux) au niveau de la deuxième dérivée, l'accélération. On peut aller plus loin encore, et garantir une constante au niveau de la troisième dérivée. On parle alors de "jerk constant".

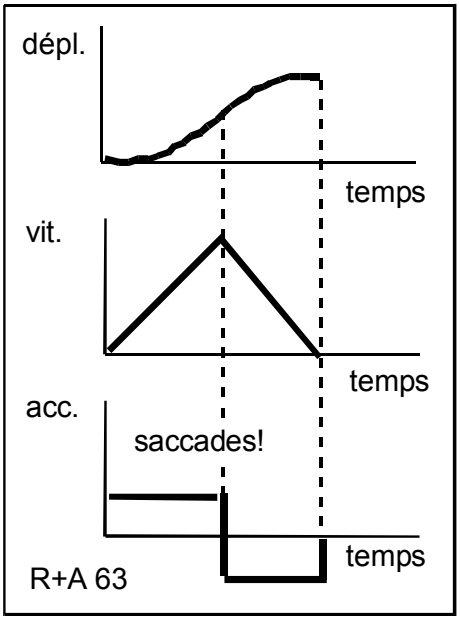


Fig. 3.3.4 Loi de mouvement à accélération constante

La douceur de la commande n'est pas sans effet sur les vibrations, et donc les erreurs de dépassement, d'une articulation imparfaitement rigide. On peut ainsi estimer que l'erreur de dépassement, D, respecte l'équation suivante, pour le cas de l'accélération constante:

$$D \le \frac{1}{\omega_0^2} \sum_{1}^{n} \sigma_i = \frac{16\,A}{\omega_0^2\ T^2}$$

PROFIL TRAPEZOÏDAL

Il est très courant en pratique de modéliser les mouvements articulaires, par rapport au temps, comme ayant des phases de démarrage et de freinage à accélération (et décélération) constante, séparées par un temps de durée variable où l'articulation se déplace à vitesse constante. Durant les phases de démarrage et de freinage, puisque l'accélération est constante, la vitesse croît et décroît de

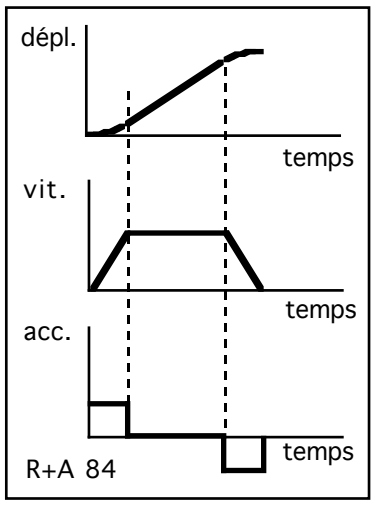


Fig. 3.3.5 Loi de mouvement "à profil trapézoïdal"

façon linéaire. Globalement, le profil de vitesse apparaît donc, en général, comme de forme trapézoïdale. Il peut arriver que le déplacement soit si court que l'axe n'a pas le temps d'atteindre la vitesse maximale. La courbe de vitesse est alors, exceptionnellement, triangulaire.

En général, on considère que le temps de freinage, t_{frein} , est égal au temps d'accélération, t_{acc}.

Les relations entre temps d'accélération, vitesse maximale, et accélération maximale sont évidentes:

$$\Rightarrow a = \frac{v_{max}}{t_{acc}} \quad \text{ou encore } t_{acc} = \frac{v_{max}}{a_{(max)}}$$

Une variante un petit peu plus compliquée du mouvement à profil trapézoïdal est celui où le freinage se fait en deux étapes: d'abord, le freinage est maximal, comme nous l'avons vu; mais avant l'arrêt total du mouvement, l'accélération est fortement réduite, passant à un palier inférieur afin de garantir une arrivée en douceur, et évitant par là dépassements et vibrations. Ceci se fait par contre au détriment du temps de cycle.

Un raffinement vise à obtenir une commande plus douce encore que celle à accélération constante. Il s'agit d'une commande continue (par morceaux) au niveau de la dérivée de l'accélération, c'est à dire au niveau du « jerk ».

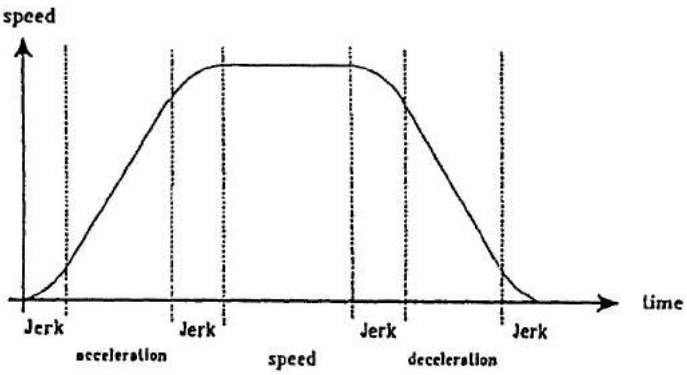


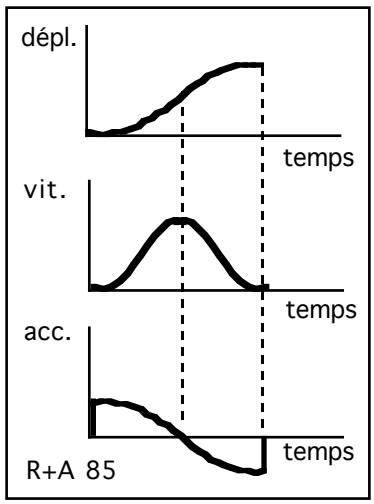
Fig 3.3.5bLoi de mouvement "à profil trapézoïdal", avec jerk constant.

En particulier, pour les robots Demaurex (Delta), la portion ordinairement constante de a(t) durant les phases d'accélération et de décélération du mouvement trapézoidal en vitesse sont eux-mêmes trapézoïdaux, ce qui implique des impulsions d'amplitude constante au niveau du jerk. Le résultat sur la trajectoire c'est l'apparition de segments de cubiques (triple intégration du jerk), qui donne l'impression sur la courbe de vitesse (v(t)) d'un trapèze à transitions arrondies (cf. fig. 3.3.5b).

C MOUVEMENTS TABULES

Alors que dans les cas précédents les mouvements pouvaient se calculer en temps réel, nous allons considérer ici la possibilité de précalculer des lois de mouvements compliquées, sous forme normalisée (normée) en amplitude et en durée. Hélas ce n'est pas toujours possible. Ainsi précisément le profil trapézoïdal que nous venons de voir change de forme, passant par exemple du triangle à un trapèze ou même à un quasi-rectangle, lorsque les phases d'accélération et de freinage sont courtes par rapport à la durée du trajet à vitesse constante.

LOI EN COSINUS



$$x(t) = \frac{A}{2}\left(1 - \cos\left(\frac{\pi}{T}t\right)\right)$$

$$v(t) = \frac{A}{2} \bullet \frac{\pi}{T} \bullet \sin\left(\frac{\pi}{T}t\right)$$

$$a(t) = \frac{A}{2} \bullet \left(\frac{\pi}{T}\right)^2 \bullet \cos\left(\frac{\pi}{T}t\right)$$

Fig. 3.3.6 Loi d'interpolation en cosinus

Nous avons vu que la douceur d'une commande variait en fonction de l'ordre de dérivation où l'amplitude était discontinue. Les fonctions trigonométriques (sinus et cosinus) ont la propriété intéressante de n'avoir jamais de dérivée discontinue. On peut donc être tenté par une loi d'interpolation temporelle en forme de cosinus.

Il faut bien voir cependant que notre loi d'interpolation n'a pas une durée infinie. Elle commence au début du mouvement et s'interrompt à la fin. Ainsi nous n'avons qu'une demi-solution: Le passage de l'accélération au freinage est infiniment doux (dans le sens que toutes les dérivées sont continues), mais par contre le début et la fin du mouvement ne se distingue guère du cas de l'accélération constante (discontinuités à la deuxième dérivée).

Le cosinus est tabulé, avec une durée de 1 s, et une amplitude de 1 (mm ou degré). Comment passer aux termes effectifs A et T?. A nous est fixé par le niveau P. C'est l'intervalle entre deux consignes successives, A_1 et A_2: $A = A_2 - A_1$. Quant à la durée, T, elle peut être tirée de A, du dépassement toléré, D, et de la fréquence propre du système entraîné, f_0:

$$D < \frac{1}{\omega_0^2} \sum_1^n \sigma_i \,, \quad \text{avec} \sum_1^n \sigma_i = 2\,\frac{A}{2}\left(\frac{\pi}{T}\right)^2$$

$$\Rightarrow D \sim \frac{1}{\omega_o^2} \cdot \frac{\pi^2 A}{T^2} \quad \text{et} \quad T = \frac{1}{2\ fo}\sqrt{\frac{A}{D}}$$

CAME ELECTRONIQUE

La méthode du profil tabulé revient à définir sous forme électronique les cames traditionnelles. On peut alors, dans la phase de conception de la commande, engager des ressources considérables afin de calculer un profil "optimal". L'utilisation de la table, lors du contrôle en temps réel, est par contre très simple et prend un temps négligeable.

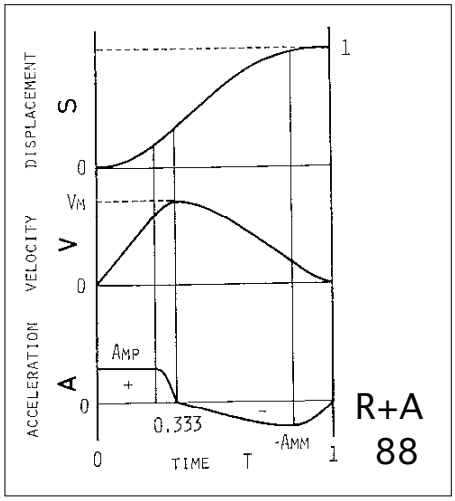


Fig. 3.3.7 Loi de mouvement "NC2" définie pour un prototype SCARA

Le profil de la fig. ci-dessus a été proposé pour la commande d'un robot de type SCARA. Nous constatons que pour le démarrage, l'accélération est énergique. Pour le freinage en revanche, le comportement est beaucoup plus doux afin d'éviter les oscillations et le dépassement. L'inversion du signe de l'accélération se fait au tiers du mouvement, ce qui limite l'excitation des vibrations aux basses fréquences, où l'atténuation est généralement moindre (peu d'harmoniques d'ordre 3).

Il n'est pas nécessaire de mémoriser à la fois les courbes de déplacement, de vitesse et d'accélération, car la courbe de déplacement est mémorisée contient sous forme implicite tous les profils dérivés (vitesse, accélération, jerk...).

Lorsqu'on compare les lois de déplacement présentées jusqu'ici, il apparaît que plus la loi de commande est douce, et plus le démarrage et le freinage se font à petite pente. Par contre, à durée de déplacement égale, une commande douce devra atteindre une pointe de vitesse élevée à mi-parcours.

On observe également que la loi en cosinus ressemble beaucoup à celle de l'accélération constante, bien qu'elle lui soit légèrement inférieure.

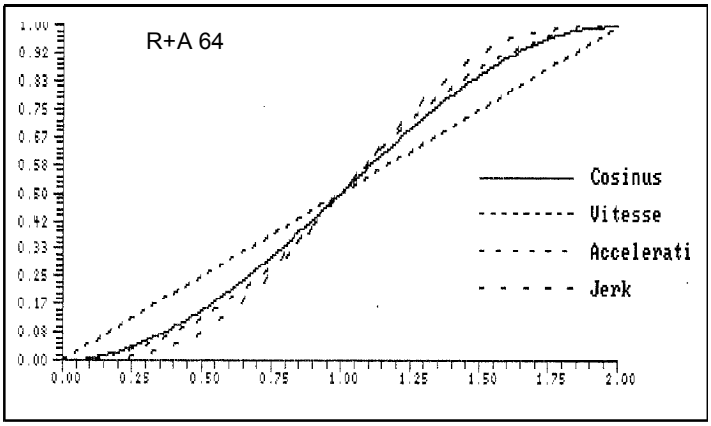


Fig. 3.3.8 Profils tabulés pour les lois à vitesse constante, à accélération constante, à jerk constant, et en cosinus

3.3.3 COORDINATION POUR COMMANDE DE TRAJECTOIRES. INTERPOLATION DANS L'ESPACE.

La coordination des articulations permet de générer différents types de trajectoires dans l'espace, linéaires, circulaires ou autres encore. Voyons d'abord le principe de la commande de trajectoire.

A PRINCIPE

Il s'agit ici de commander des trajectoires dans l'espace. La première question est celle du choix des coordonnées. L'interpolation elle-même est presque toujours linéaire. Mais veut-on cette interpolation linéaire sur les coordonnées articulaires, les coordonnées cartésiennes de l'atelier, ou encore en coordonnées cylindriques ou polaires? Les deux premiers cas sont traités de façon optimisée dans la plupart des commandes de robots. Le dernier par contre, est souvent laissé aux soins de l'utilisateur, qui peut le réaliser en mode "procédural" que nous verrons plus loin.

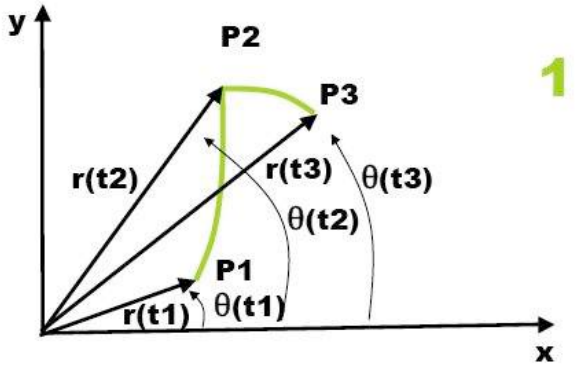


Fig. 3.3.9 Trajectoire définie entre P_1, P_2, et P_3, par interpolation linéaire dans l'espace des articulations. La trajectoire est vue dans l'atelier. Comme le robot est cylindrique, la trajectoire est faite de segments de spirale.

Il est indéniable que le mot "linéaire" qualifiant la méthode d'interpolation adoptée conduit souvent à des malentendus , bien qu'il soit exact, car on l'applique à des coordonnées différentes de cas en cas (articulaires, spatiales, temporelle...). On peut tenter de décrire les différents modes d'interpolation linéaire par des termes peut-être techniquement discutables, mais intuitivement plus parlant: dans le cas de l'interpolation linéaire dans l'espace des articulations, on conviendra de parler de "mouvement

proportionnel"; si elle s'effectue en coordonnées spatiales (coordonnées de l'atelier ou du préhenseur) on parlera de "mouvement cartésien".

L'évolution par rapport au temps des consignes de position ne nous intéresse plus directement, car elle n'affecte pas la forme de la trajectoire, en mouvement normalement coordonné. C'est indirectement, à cause d'un nécessaire synchronisme entre les différentes coordonnées, que l'on en tiendra encore compte.

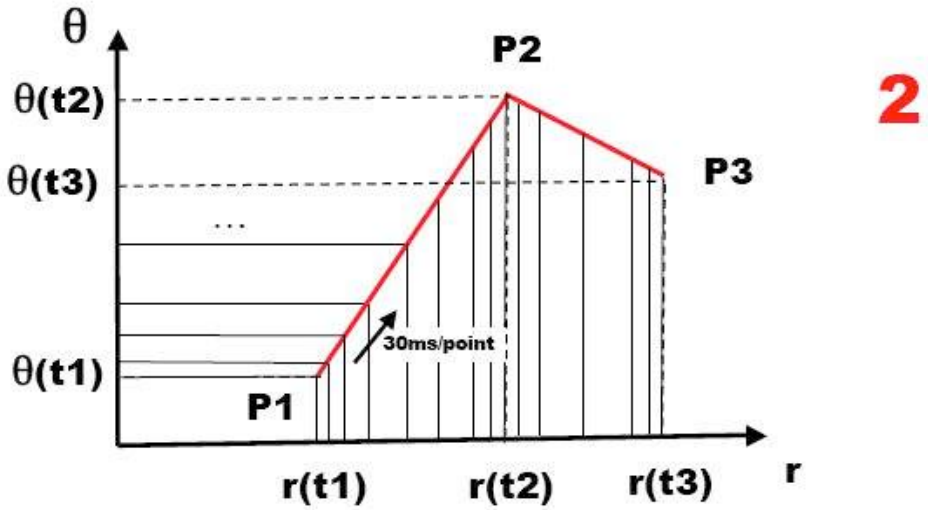


Fig. 3.3.10 Même trajectoire que dans la fig. précédente, mais observée cette fois dans l'espace des coordonnées articulaires. On observe également les points correspondant à l'envoi de consignes aux servocommandes. Ils sont rapprochés lorsque le robot se déplace à basse vitesse, car ils se calculent à intervalles de temps réguliers.

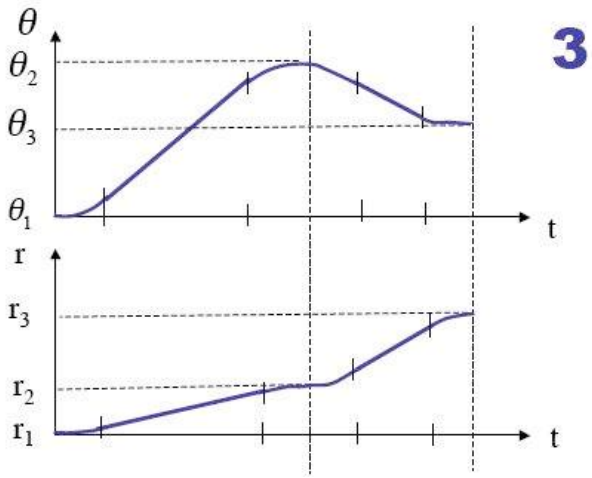


Fig. 3.3.11 Même trajectoire que dans les fig. précédentes, mais observée cette fois dans l'espace du temps, et pour chacune des articulations.

Dans une première étape, on choisit une durée de mouvement, T, commune pour toutes les coordonnées. Cette durée est la plus petite possible, c'est-à-dire la plus petite que toutes les coordonnées peuvent respecter. Une coordonnée lente jouera un rôle important dans le choix de T.

Mais ce peut aussi en être une autre, qui, bien que rapide, prend beaucoup de temps car elle doit faire un très grand déplacement par rapport à son axe propre.

Une fois T fixé, on peut pour chaque coordonnée interpoler par rapport au temps l'amplitude des consignes définissant la trajectoire. Si ces consignes s'expriment en coordonnées articulaires, on pourra les envoyer aux servocommandes. Sinon, il conviendra encore de les convertir.

B MOUVEMENT LINEAIRE DANS L'ESPACE DES ARTICULATIONS

La situation la plus courante est celle où l'interpolation se fait linéairement dans l'espace des coordonnées articulaires. Si les consignes nous parviennent du niveau P dans ces coordonnées là, on peut sans autre passer à l'interpolation et transmettre les consignes interpolées au niveau S.

Si par contre, ce qui est courant également, les consignes nous parviennent du niveau plus élevé en coordonnées spatiales (cartésiennes, dans l'atelier), il faudra tout d'abord les convertir en coordonnées articulaires à l'aide de la solution cinématique inverse.

Considérons l'interpolation entre deux points P_i et P_{i+1}.

$$\vec{P_i} = \begin{pmatrix} \theta_{1i} \\ \theta_{2i} \\ \theta_{...i} \\ \theta_{ni} \end{pmatrix} \qquad \vec{P_{i+1}} = \begin{pmatrix} \theta_{1\ i+1} \\ \theta_{2\ i+1} \\ \theta_{...\ i+1} \\ \theta_{n\ i+1} \end{pmatrix}$$

On a déterminé que l'articulation qui nécessite le plus de temps est l'articulation θ_j, qui va de l'amplitude θ_{ji} à θ_{ji+1} en suivant la loi de mouvement f(t) (cf. § 3.3.2). Nous avons donc:

$\theta_j(t) = f(t)$, dans l'intervalle t_i à t_{i+1}

Nous sommes dans l'espace des variables articulaires, aussi le temps ne nous intéresse pas directement. Nous décidons que toutes les coordonnées soient liées l'une à l'autre par une loi linéaire. En particulier il en résulte que toutes les articulations peuvent se connaître par rapport à la variable interpolée par rapport au temps, θ_j :

$\theta_k = a_k (\theta_j - \theta_{ji}) + \theta_{k\ i}$, où k désigne chacune des n variables articulaires

Les coefficients a_k sont implicitement imposés par les consignes P_i et P_{i+1}. Leur valeur est la suivante:

$$a_k = \frac{\theta_{k\ i+1} - \theta_{k\ i}}{\theta_{j\ i+1} - \theta_{j\ i}}$$

On peut théoriquement se passer du temps, mais pratiquement on va cependant l'utiliser afin de paramétrer la droite correspondant au mouvement:

$\theta_k(t) = a_k (\theta_j (t) - \theta_{ji}) + \theta_{k\,i}$, dans l'intervalle t_i à t_{i+1}

EXEMPLE D'INTERPOLATION LINEAIRE

Considérons le mécanisme de la fig. 3.3.9. Les coordonnées articulaires sont θ et r. Le mouvement demandé va faire passer d'un point P_1 à un autre point P_2.

$$P_1 = \begin{pmatrix} \theta_1 \\ r_1 \end{pmatrix} = \begin{pmatrix} 30 \\ 12 \end{pmatrix} \qquad P_2 = \begin{pmatrix} \theta_2 \\ r_2 \end{pmatrix} = \begin{pmatrix} 60 \\ 8 \end{pmatrix}$$

Les déplacements A_θ et A_r valent respectivement:

$$A_\theta = (\theta_2 - \theta_1) = (60 - 30) \qquad A_r = (r_2 - r_1) = (8 - 12)$$

Considérons que les vitesses maximales en rotation et en déplacement radial sont respectivement de 180°/s et de 50 cm/s. On en tire le temps minimal de déplacement, pour chacune des articulations, sous l'hypothèse de loi de mouvement à vitesse constante, de 30/180 s, et de 4/50 s respectivement.

Le temps le plus long étant de 0.17 s, dû au grand déplacement angulaire, on l'impose également aux autres articulations, c'est-à-dire, dans notre cas, à l'articulation radiale.

Dès lors, le déplacement peut être calculé avec une loi de mouvement quelconque, par exemple trapézoïdale, pour l'une des articulations (par exemple θ). On obtient à chaque instant pour l'autre (la coordonnée radiale) la consigne adéquate par fonction linéaire:

$$r(t) = r_1 + \left(\frac{r_2 - r_1}{\theta_2 - \theta_1}\right)(\theta(t) - \theta_1) = 12 + \left(\frac{8-12}{60-30}\right)(\theta(t) - 30) = -\frac{2}{15}\theta(t) + 16$$

C MOUVEMENT LINEAIRE DANS L'ATELIER

Alors que précédemment la coordination était assurée par une interpolation linéaire en espace articulaire des consignes provenant du niveau P, il s'agit ici d'appliquer le même type d'interpolation aux coordonnées spatiales de l'atelier (coordonnées cartésiennes).

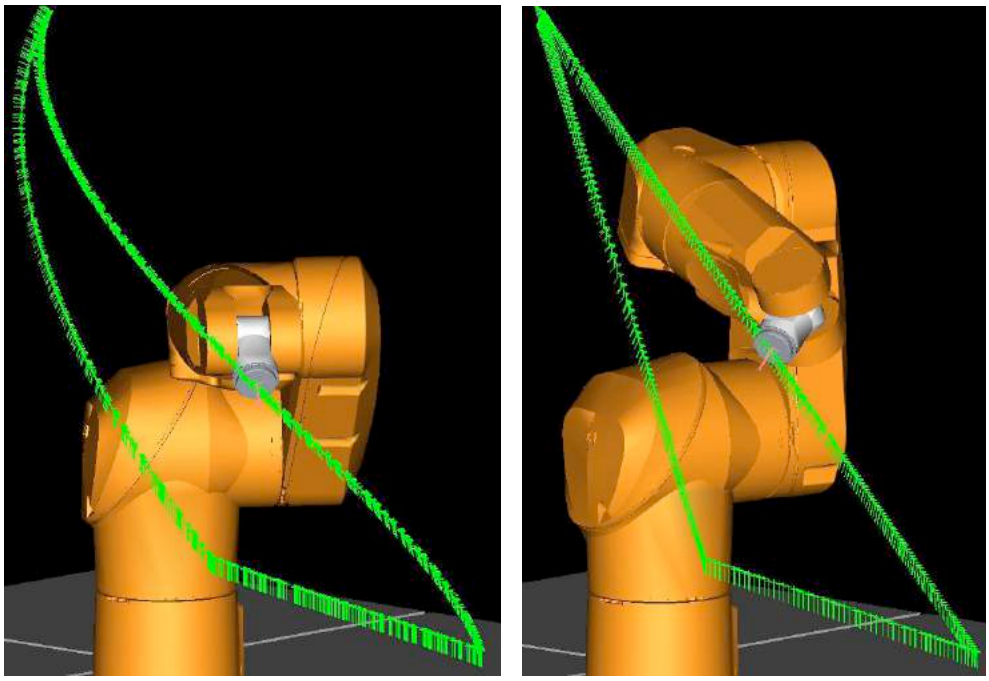


Fig. 3.3.11b Mouvements en mode « articulaire » (cf. mouvement linéaire dans l'espace des articulations), à gauche, et en mode « linéaire » dans l'atelier (à droite). Dans les deux cas, les mêmes trois points sont définis, et les trois segments de trajectoire s'enchaînent « point-à-point » (Illustrations de Sarah Pitteloud et Adrien Villoz, « Stäubli TX-40 », rapport HEIG-VD.LaRA, 29.01.2014).

L'avantage de donner le mouvement dans un espace cartésien plutôt que de le donner dans l'espace des articulations, est que le mouvement est bien défini, et qu'il peut être décrit par une suite de segments depuis le point de départ jusqu'au point d'arrivée.

L'interpolation des coordonnées spatiales ne concerne pas uniquement les déplacements en translation, mais touche également la gestion de l'orientation du préhenseur. Alors que dans le premier cas, le résultat est évident, trajectoire rectiligne dans l'atelier, dans le deuxième, c'est en particulier le problème de la rotation du préhenseur autour d'un point librement défini qu'il faut résoudre. Du point de vue des calculs, les deux aspects se confondent.

Le *calcul* de trajectoires linéaires en coordonnées cartésiennes est généralement difficile. Ceci est étonnant pour le non-spécialiste car nous avons tous une expérience importante dans le maniement de nos propres bras, et cela semble aller de soi.

Pour calculer un mouvement linéaire dans l'atelier, il faut appliquer les lois de mouvement aux coordonnées spatiales. Puis pour chaque valeur interpolée, au rythme d'environ 40 valeurs par secondes, il faut évaluer la solution cinématique inverse, qui, nous l'avons vu dans la partie 2, nous convertit les coordonnées spatiales en coordonnées articulaires.

La solution cinématique inverse, pour un robot, comprend généralement beaucoup de termes trigonométriques, ce qui rend énorme la masse de calculs à évaluer. Une alternative partielle consiste à utiliser, par segments relativement grands de la trajectoire, des relations linéaires entre variables spatiales et variables articulaires, le jacobien. Ces relations s'établissent par calcul différentiel.

Un autre problème du mouvement cartésien, c'est celui dérivant d'une connaissance imparfaite des éléments géométriques du manipulateur. La solution cinématique inverse implique des termes jugés constants (paramètres D-H, tels que par exemple l'angle formé par deux axes successifs de mouvement) qui en fait varient en fonction des tolérances d'usinage, mais aussi avec le vieillissement et les charges dynamiques instantanées. En valeur absolue, les erreurs de positionnement peuvent être très grandes (10 à 100 fois l'erreur nominale), mais en terme relatif (cf. notion de répétabilité introduite dans la partie 1) ceci est acceptable. Ceci est d'autant plus vrai que le manipulateur ne travaille pas dans un hypothétique référentiel absolu, mais ce qui est déterminant pour lui c'est de ce positionner de façon précise par rapport aux objets qui l'entourent. Dans ce contexte, la facilité de gestion de trajectoires cartésiennes, impliquant un mode agréable de dialogue avec le programmeur et les capteurs extéroceptifs s'avère déterminante.

Enfin, une difficulté encore propre au mouvement cartésien, c'est qu'il masque à l'utilisateur le travail effectif de chaque articulation. Cela a deux conséquences principales. D'une part, le déplacement des articulations nécessaires au mouvement cartésien n'est parfois pas possible; l'une des articulations est en butée, ou plusieurs axes de mouvements sont alignés: la solution cinématique inverse n'a alors pas de solutions; cela implique un arrêt du manipulateur. D'autre part des vitesses même modestes sur les coordonnées spatiales impliquent parfois de très grande vitesses articulaires. Lorsque celles-ci ne peuvent être atteintes, des erreurs grossières sont possibles, sur la trajectoire.

Dans les machines-outils, les tolérances le long des trajectoires sont très faibles, aussi les points interpolés sont très rapprochés et nombreux. Pour pouvoir exécuter toute la masse de calculs nécessaires, des circuits matériels spéciaux sont utilisés. Cependant, l'interpolation linéaire n'est alors assurée que pour 2 ou 3 coordonnées simultanément, et les coordonnées articulaires sont elles-mêmes mécaniquement linéaires (axes prismatiques) ce qui simplifie le problème. Pour les robots il y a moins de points, mais par contre ils concernent l'ensemble des articulations (typiquement, 4 à 6 ddl).

D MOUVEMENT CIRCULAIRE OU PROCEDURAL

Dans l'étage de coordination de la commande de robot, il n'est généralement prévu qu'une interpolation linéaire, soit dans l'espace des variables articulaires, soit dans celui des coordonnées spatiales (x, y, z, et angles d'Euler).

Il est néanmoins possible d'obtenir des trajectoires circulaires, sinusoïdales, ou de forme quelconque en intervenant au niveau supérieur, celui de la programmation. Pour avoir une trajectoire de qualité, il faut que les calculs se fassent à un rythme suffisant (40 points/s). Entre les points calculés, on peut schématiquement admettre que l'unité de coordination génère des segments de droite. Mais en fait l'approximation peut être meilleure encore si l'on travaille en mode dit "à trajectoire continue".

3.3.4 COMMANDE POINT PAR POINT ET TRAJECTOIRE CONTINUE

Jusqu'ici nous avons considéré des lois de mouvements pour un déplacement élémentaire, entre le point courant, P_i et la consigne suivante P_{i+1}. Au point de départ comme à l'arrivée, le bras est à l'arrêt. On parle dans ce cas d'une commande qui travaille en mode "point-à-point".

Lorsque plusieurs consignes se succèdent, en provenance du niveau P, il est possible et avantageux d'optimiser les transitions. Si par exemple les consignes successives définissent une droite, il vaut mieux ni s'arrêter, ni même ralentir au voisinage des points de consigne. Le type de commande qui optimise ainsi le passage d'un segment de trajectoire au suivant (intervalles P_{i-1} à P_i, et P_i à P_{i+1}) est dit à "trajectoire continue".

Voyons d'abord les avantages et défauts respectifs de ces deux types de commande. Nous examinerons ensuite en détail l'approximation polynomiale inhérente aux trajectoires continues.

A COMMANDE POINT A POINT

La commande point à point a pour elle l'avantage de la simplicité de l'algorithme de commande. Elle présente aussi l'avantage parfois déterminant de passer exactement par les points de consignes reçus du niveau P.

Par contre, il existe deux inconvénients majeurs propres à ce mode de commande.

Temps de cycle. Le temps de cycle est relativement élevé, car même si les points successifs sont alignés, la commande bloque le bras au passage de chacun des points. Si la distance est grande entre consignes successives, ce problème n'est pas sérieux. Mais s'ils sont rapprochés, comme cela est presque obligatoirement le cas lors de la phase finale des mouvements d'assemblage, ou lorsque l'on travaille en mouvement procédural, ce défaut peut être extrêmement gênant.

Vibrations. Les successions d'accélération et de décélération, inévitables en mouvement point-à-point sont aussi causes de problèmes, surtout lorsque leur alternance est rapide, dans les petits

mouvements ou en mouvement procédural. Les conséquences fâcheuses possibles sont multiples: risque de perte de la charge, usure prématurée de la mécanique, oscillations et erreurs le long de la trajectoire.

B TRAJECTOIRE CONTINUE

Nous allons maintenant traiter le cas de la continuité de la trajectoire que les points de consigne provenant du niveau P définissent.

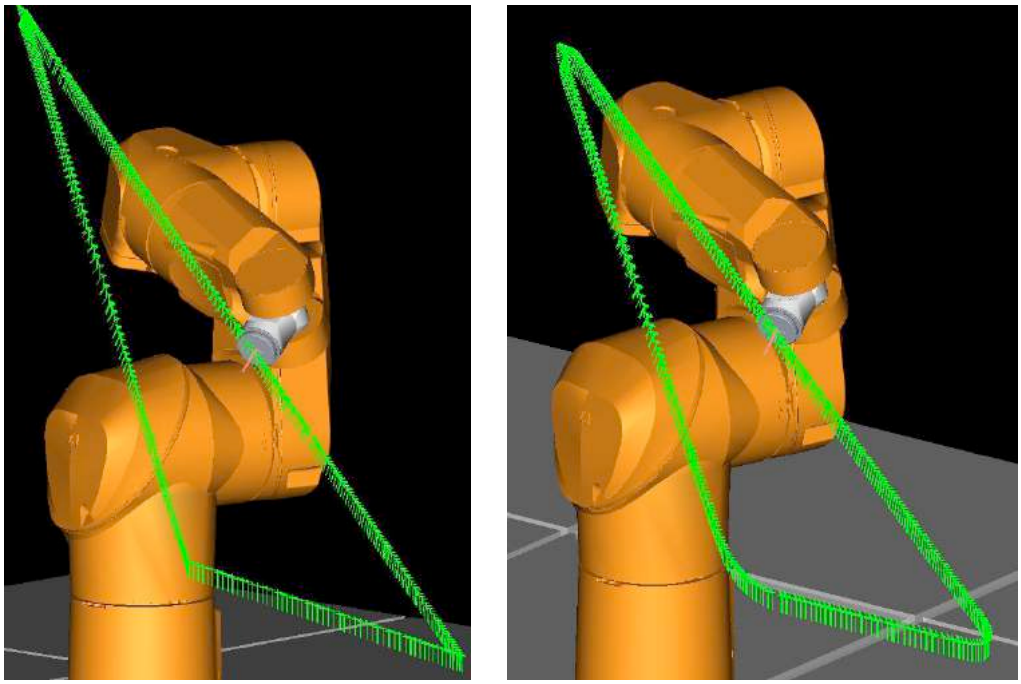


Fig. 3.3.11c Mouvements en mode « point-à-point» (arrêt en chaque point), à gauche ; et en mode « continu», avec 50 mm de zone de lissage (à droite). Dans les deux cas, les mêmes trois points sont définis, et les trois segments de trajectoire s'enchaînent en mode « linéaire » (Illustrations de Sarah Pitteloud et Adrien Villoz, « Stäubli TX-40 », rapport HEIG-VD.LaRA, 29.01.2014).

L'avantage d'un tel type de commande est la diminution du temps de cycle, ainsi que le "coulé" du mouvement, qui se traduit par des saccades et vibrations moindres.

Il faut bien voir cependant que cette solution est susceptibles d'erreurs locales parfois inadmissibles. Paradoxalement, elle est précise *entre* les points de consignes, mais c'est dans leur voisinage immédiat que les erreurs les plus grandes se manifestent. Cet effet se comprend mieux sur un graphique (cf. fig. 3.3.11).

Le mode de commande à trajectoire continue n'est donc acceptable que si une marge d'erreur est tolérée. Comme ce n'est pas toujours le cas, le programmeur doit de façon explicite indiquer lorsqu'il la désire, ou quand au contraire, il convient d'y faire exception.

Un désavantage occasionnel, lié au principe même de ce mode de commande, est que l'interprétation du programme doit précéder de plusieurs instructions (au moins jusqu'à l'instruction donnant la consigne suivante) l'exécution, par le bras, du programme. Il faudra donc prendre garde à la synchronisation du mouvement avec les capteurs et autres équipements périrobotiques gérés simultanément par le programme. Le cas échéant, il faut bloquer l'exécution du programme jusqu'à ce que le mouvement en cours soit terminé.

C APPROXIMATION POLYNOMIALE

Voyons maintenant comment une trajectoire continue se calcule. On peut distinguer deux étapes principales. Dans la première, une trajectoire "exacte", mais non continue en vitesse s'élabore, sur la base des consignes transmises par le niveau P. Ensuite, dans une démarche qui rappelle un peu celle adoptée pour le déplacement à profil trapézoïdal de vitesse, le passage entre segments de trajectoire à vitesse constante est assuré, par une accélération douce, au détriment éventuel d'une légère erreur sur la position.

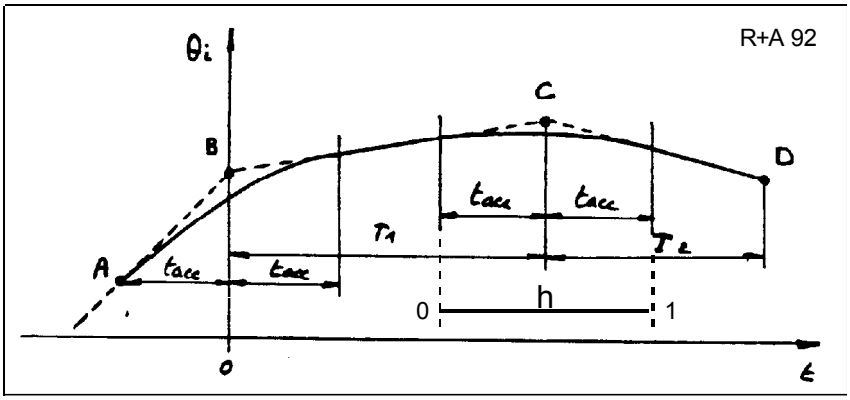


Fig. 3.3.12 Approximation polynomiale de la trajectoire commandée, au voisinage des points de consigne B et C. Une seule coordonnée, θ_i, est représentée ici.

La trajectoire "exacte". Dans une première étape, il s'agit à partir des consignes successives P_i reçues du niveau P, de générer une esquisse de trajectoire.

Seul l'ordre des consignes est explicitement déterminé par le niveau P. Eventuellement, des paramètres de vitesse et d'accélération maximales sont données par ailleurs, mais en aucun cas le programme ne définit l'intervalle de temps devant séparer le passage par deux points successifs.

On démarre tout simplement en considérant que le déplacement d'un point au suivant se fait à vitesse constante. Comme on l'a déjà dit, ce mouvement n'est pas réaliste dans la mesure où il implique des pointes infinies d'accélération. Néanmoins c'est une approche utile à ce stade.

Comme toujours en coordination, il faut déterminer le temps T_{ij} nécessaire à chacune des coordonnées (indice j), pour le déplacement dans les intervalles successifs, P_i à P_{i+1}. Dans chaque intervalle (pour chaque valeur de i), le temps le plus long (maximum des T_{ij}) est retenu pour

l'ensemble des articulations, c'est-à-dire pour la trajectoire elle-même. Un exemple de ce type de coordination a été développé au §3.3.3.B. La fig. 3.3.11 illustre aussi ce calcul. On y observe qu'entre les consignes B et C, puis C et D, les durées T_1 et T_2 ont été déterminées. Il y apparaît également que la vitesse, pour la coordonnée représentée sur la figure, est inférieure pour ces deux segments à ce qu'elle valait entre A et B. On en conclut qu'elle "attend" vraisemblablement une autre coordonnée travaillant à vitesse limite.

Les changements de vitesse. Il faut maintenant remplacer les points de discontinuités (en vitesse) de la trajectoire esquissée jusqu'ici.

Un certain temps est nécessaire aux articulations pour qu'elles puissent changer de vitesse, puisque les accélérations sont forcément limitées par la capacité des actionneurs. Dans le pire des cas, la vitesse initiale est maximale, et il faut non seulement la réduire à zéro, mais encore atteindre la vitesse maximale en sens inverse. Appelons ce temps maximal d'accélération t_{acc}, et admettons qu'il est également applicable au freinage. Par prudence, nous adopterons cette valeur pour toutes les transitions. L'approximation va donc se concentrer sur une période de 2 t_{acc}, qui est centrée sur chaque point de consigne (point de discontinuité de la vitesse).

Pour alléger les notations, substituons à la variable temporelle t une autre variable, h. Alors que la première évolue, pour la transition au voisinage de P_i, entre $t_i - t_{acc}$, et $t_i + t_{acc}$, nous définissons h de manière à ce qu'elle passe, pour le même bout de trajectoire, de 0 à 1:

$$t_i = \sum_{k=1}^{i} T_k$$

$$h = \frac{t - t_i + t_{acc}}{2\ t_{acc}}$$

La fonction d'approximation à définir pour le passage d'un segment de trajectoire au suivant, c'est-à-dire dans le domaine de h allant de 0 à 1 doit satisfaire à un certain nombre de conditions.

Il faut tout d'abord respecter une contrainte de position, C_1. Mais il faut en plus générer une courbe douce, sans choc ni saccade, ce qui se traduit par des contraintes C_2 à C_5 de continuité en vitesse et en accélération, aux deux extrémités de la zone d'approximation. Au total nous avons 5 contraintes, ce qui est compatible avec un polynôme du $4^{ième}$ degré. Le polynôme apparaît à l'équation E.

Voyons en détail l'approximation de la trajectoire de la fig. 3.3.11 au voisinage du point B.

$$C_1: \quad \theta\ (h=0) = B + (T_1 - t_{acc})\ \frac{C-B}{T_1}$$

$$C_2: \quad \dot{\theta}\ (h=0) = \frac{C-B}{T_1}$$

$$C_3: \quad \dot{\theta}\ (h=1) = \frac{D-C}{T_2}$$

C_4: $\ddot{\theta}(h=0) = 0$

C_5: $\ddot{\theta}(h=1) = 0$

E: $\theta(h) = a_4h^4 + a_3h^3 + a_2h^2 + a_1h + a_0$

Le polynôme en h doit satisfaire aux 5 contraintes sus-mentionnées. On peut donc en évaluer les coefficients a_i:

C_1 et E $\Rightarrow$ $a_0 = B + (T_1 - t_{acc}) \frac{C-B}{T_1}$

C_2 et E $\Rightarrow$ $a_1 = \frac{C-B}{T_1}$

C_4 et E $\Rightarrow$ $a_2 = 0$

C_3 et E $\Rightarrow$ $\dot{\theta}(h=1) = \frac{D-C}{T_2} = 4\,a_4h^3 + 3\,a_3h^2 + a_1$

$\Rightarrow E_2$: $4\,a_4 + 3\,a_3 = \frac{D-C}{T_2} - \frac{C-B}{T_1}$

C_5 et E $\Rightarrow$ $\ddot{\theta}(h=1) = 0 = 12\,a_4 + 6\,a_3$

$\Rightarrow E_3$: $a_3 = -2\,a_4$

E_3 et $E_2 \Rightarrow$ $4\,a_4 - 6\,a_4 = \frac{D-C}{T_2} - \frac{C-B}{T_1}$

$\Rightarrow$ $a_4 = -\frac{1}{2}\left(\frac{D-C}{T_2} - \frac{C-B}{T_1}\right)$

L'approximation faite ci-dessus n'est pas fondamentalement différente de celle opérées par les fonctions de Bézier (spline B) en infographie ou en modélisation géométrique.

EXERCICE

Considérons l'exemple de trajectoire interpolée entre les points A et D de la fig. 3.3.11, par des polynômes de degré 4.

- Quelle est l'erreur de position au voisinage de C (en T_1) ?
- Que vaut la vitesse en ce point (en T_1) ?

3.3.5 MODIFICATION DE TRAJECTOIRE

Jusqu'ici nous avons vu que par programmation, et en tenant compte de capteurs extéroceptifs, il était possible de déplacer le préhenseur selon des trajectoires variées. Cependant, cela se fait au niveau P, qui se caractérise par une grande flexibilité, mais aussi par des cadences d'exécution d'instructions relativement faibles.

A l'inverse, les constantes de temps sont très petites au niveau des servocommandes (niveau S), mais les consignes doivent obligatoirement y être fournies en coordonnées articulaires, ce qui est une limite déterminante, car les capteurs extéroceptifs ne fonctionnent pratiquement jamais selon des axes de mouvements articulaires.

Pour les meilleures performances, il faut intervenir dans l'étage de coordination. De toute façon, la conversion de coordonnées spatiales en coordonnées articulaires s'y déroule, et de plus, les routines d'interpolation sont optimisées pour travailler vite. Lorsque l'étage de coordination permet de prendre en compte, à sa cadence relativement élevée (typiquement 40 points par seconde), des données extérieures de corrections de déplacement, on parle de "modification de trajectoire".

Les signaux reçus servent à définir un déplacement instantané relatif à la trajectoire nominale commandée par le niveau P. Du point de vue des calculs, nous avons maintenant en particulier un produit matriciel supplémentaire, qui vient s'intercaler entre la procédure d'interpolation linéaire en coordonnées spatiales, et l'opérateur cinématique inverse:

A: $T'(t) = T_{nominale}(t) \cdot T_{modif}(t)$

B: $T'(t) = T_{modif}(t) \cdot T_{nominale}(t)$

La première équation est applicable si les corrections s'interprètent dans le repère du préhenseur, alors que l'équation B s'utilise lorsque les corrections externes sont données par rapport au repère de la base du robot, soit, pratiquement dans les coordonnées d'atelier.

L'utilité d'une commande de robot permettant la modification de trajectoire en temps réel n'est pas négligeable. Elle permet par exemple de déposer des colles à distance constante du bord des pièces, d'ébavurer des objets en leur applicant une force constante, normale à la surface de contact. Une autre classe d'applications intéressantes est celle où les équipements périrobotiques, ou la pièce à travailler, sont en mouvement par rapport au robot; comme, par exemple, lorsque les pièces sont sur une chaîne de fabrication.

3.4 COMMANDE ET ASSERVISSEMENT D'AXES

Les systèmes d'asservissement d'axe sont tout au bas de la hiérarchie de commande. Ils travaillent soit en boucle fermée, soit souvent aussi en boucle ouverte. Ces divers points sont abordés successivement ci-dessous. Ils sont suivis d'un exemple de circuit de commande pour moteur: le HCTL-1000

3.4.1 ROLE DES SYSTEMES D'ASSERVISSEMENT D'AXE

Au bas de la hiérarchie que représente une commande de robot, ou très souvent également parmi les périphériques d'un ordinateur industriel, on rencontre des systèmes d'asservissement d'axes. On emploie aussi parfois, pour les désigner, les termes de servocommandes ou de systèmes d'entraînement d'axe.

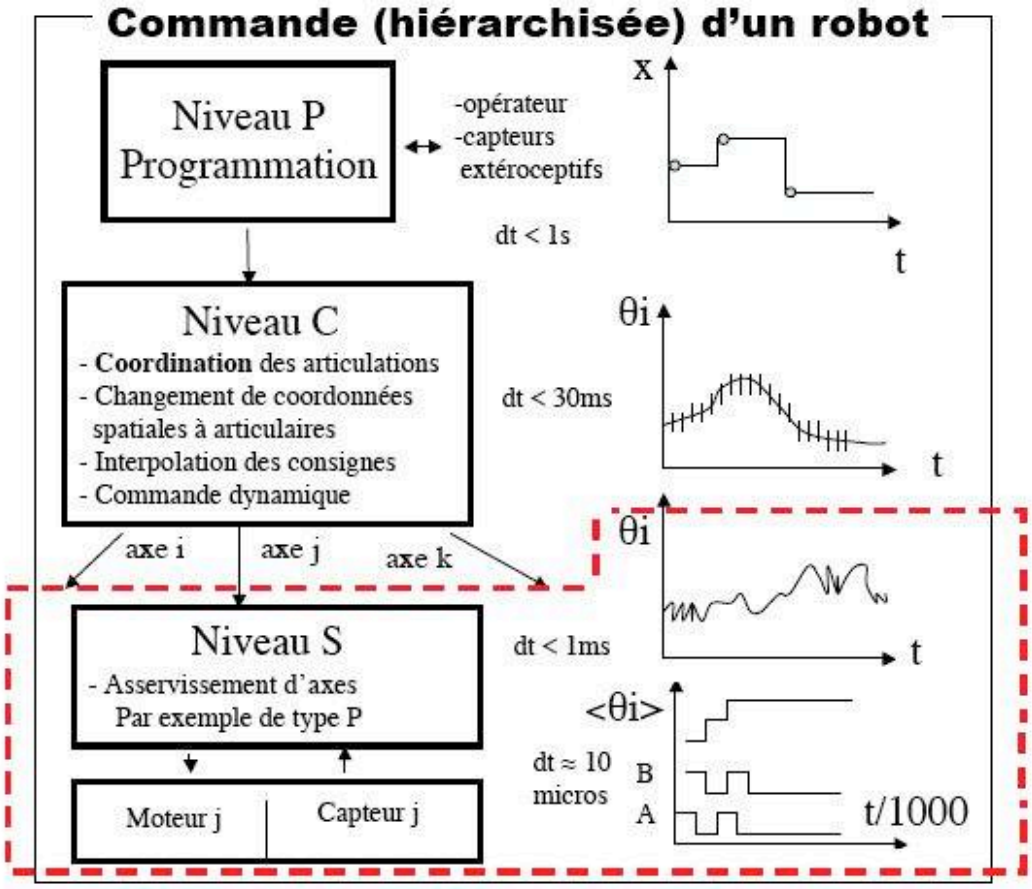


Fig. 3.4.1 Commande au niveau des articulations individuelles (cadre rouge pointillé). Une seule coordonnée, θ_i, est représentée ici. Le premier graphique montre typiquement la tension au bornes d'un moteur, et le deuxième les signaux de capteurs, directs (A, B) et intégrés ($<\theta_i>$).

Schématiquement, les servocommandes reçoivent des niveaux supérieurs une consigne, décrivant l'état désiré du système réglé. Ils envoient alors un signal de commande aux actionneurs, tout en tenant compte par des capteurs de mesure de l'état effectif du système entraîné.

On ne parlera pas, dans cette partie du cours, ni des capteurs utilisés, ni des actionneurs, dont les caractéristiques ne peuvent être ignorées pour un réglage effectif. On ne parlera pas non plus des étages de puissance qui alimentent les actionneurs. On s'en tiendra ici à une présentation des techniques fondamentales en réglage, et même cela sera fait de façon relativement superficielle.

En général, comme nous l'avons vu, la servocommande tient compte par capteurs de l'état du système. Le chemin de l'information part de la consigne, traverse le régulateur, les circuits de puissance et l'actionneur, aboutit sur le système entraîné, puis, par le capteur, revient sur le régulateur. La boucle se referme, comme nous l'avions observé dans l'une des premières figures du cours.

Mais ce n'est pas toujours le cas. Il est possible d'entraîner certains axes en boucle ouverte, c'est-à-dire sans prélever par capteur une information quelconque sur leur état effectif.

3.4.2 REGULATION EN BOUCLE FERMEE

Le réglage en boucle fermée est le thème essentiel des théories de l'automatique. Dans notre contexte, cependant, les techniques sont relativement simples, pour au moins deux raisons. D'une part, la servocommande s'intègre dans une hiérarchie de commande, remontant à des niveaux d'abstraction et de généralité qu'elle ignore, mais où des signaux de mesure adaptés sont aussi pris en compte. D'autre part, les servocommandes doivent travailler à une cadence (bande passante, pour les circuits analogiques, et fréquence d'échantillonnage pour les circuits numériques) très élevée. Les constantes de temps y sont de l'ordre de la milliseconde.

Les servocommandes ne règlent pas toujours les mêmes grandeurs. Les plus courantes sont la position, la vitesse, et le couple de l'actionneur. Le régulateur peut lui-même avoir plusieurs boucles de réglage.

Choix de type et de structure de régulation

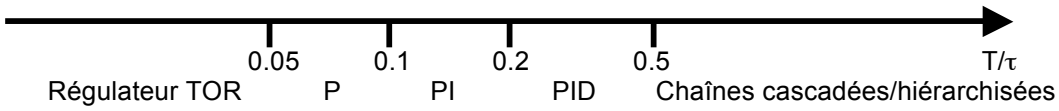
Considérons la régulation d'un système en boucle fermée. Dans les cas les plus simples une commande de type tout-ou-rien suffit. Mais pour des cas plus délicats, la commande doit se faire de façon fine. Travaillant à niveaux d'amplitude multiples, l'unité de commande prend alors l'aspect d'un régulateur bouclé de type P, PI, PD, ou PID notamment.

Enfin il subsiste encore une classe de systèmes non gérables par un seul régulateur PID, quelque soit la valeur des gains considérée. Il faut alors enrichir la structure de commande avec d'autres régulateurs assurant par ailleurs une partie du travail.

La figure ci-dessous propose, comme point de départ, un critère pour le choix du type de régulateur approprié. La variable libre est un rapport lié à la constante de temps caractéristique de la commande, T, (comprenant le temps de calcul du régulateur et la somme des éventuels retards purs présents

dans la boucle de réglage) et la constante de temps caractéristique du système réglé, τ, (somme de toutes ses constantes de temps individuelles, s'il y en a plusieurs). On définit en général l'agilité comme l'inverse du temps de réaction caractéristique (ici : agilité A_C=**1/** T ; agilité A_S=**1/** τ). L'agilité relative de la commande par rapport au système à commander vaut dès lors le rapport suivant : A_r= A_C/ A_S ce qui est équivalent à A_r= τ / T.

On observe que pour des systèmes à T petit (régulation relativement agile), les solutions simples sont appropriées. Lorsqu'au contraire T avoisine ou dépasse la constante de temps caractéristique du système, τ, des modes de régulations plus évolués doivent s'envisager.



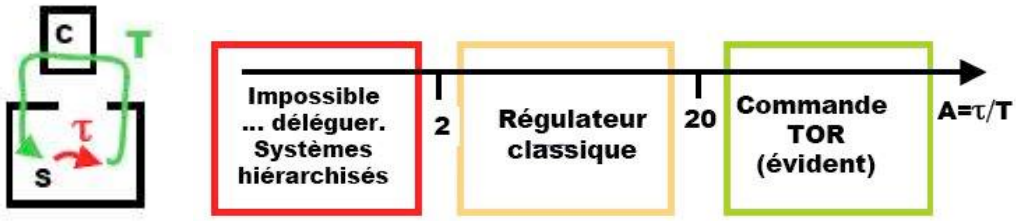


Fig. 3.4.2.1 On observe que pour des commandes, C, relativement rapides avec retards faibles (T petit, donc vitesse de réaction, ou agilité, grande, A_C=**1/** T), les solutions simples sont appropriées. Lorsqu'au contraire, T avoisine ou dépasse la constante de temps caractéristique , τ, du système à commander, S, des modes de régulation plus évolués doivent s'envisager. L'agilité *relative*, A_r, est définie comme le rapport de la seconde grandeur sur la première (A_r= A_C / A_S =τ/T)

A REGLAGE ANALOGIQUE

Le réglage analogique bénéficie d'environ un demi-siècle d'expérience. Des méthodes mathématiques ont été développées pour synthétiser des régulateurs et en analyser les performances (transformée de Laplace, lieu d'Evans, diagramme de Bode, etc.).

Le régulateur le plus typique en réglage analogique est celui que l'on appelle proportionnel-intégral-dérivé (PID). Il vaut la peine de le discuter quelque peu. Nous le faisons en privilégiant une description plus intuitive que mathématique, car il faut en sentir les aspects fondamentaux au moment de se convertir aux techniques numériques.

CONTROLE PROPORTIONNEL INTEGRAL DERIVE

A l'entrée d'un régulateur, on commence généralement par évaluer la différence instantanée entre la consigne, C et la mesure, M. Ceci nous donne l'erreur, ε.

$$\varepsilon = C-M$$

Le régulateur PID élabore un signal de commande, Y, à partir de ce signal d'erreur. Il lui fait d'abord subir trois opérations en parallèle, puis additionne les résultats dans un étage de sortie. Chacune des trois opérations correspond à l'une des 3 lettres de son nom.

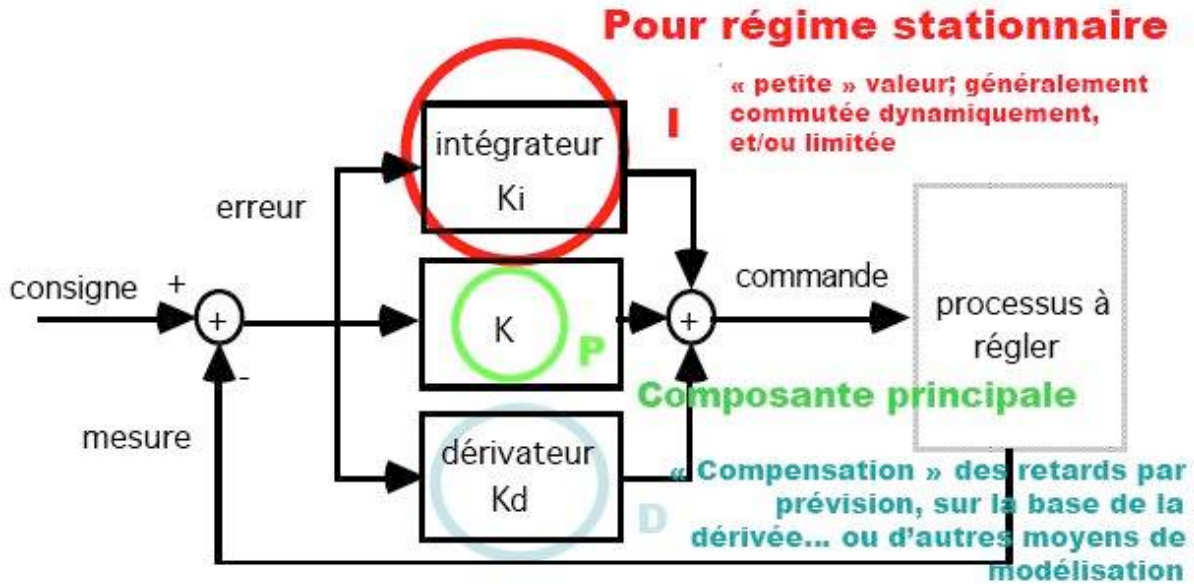


Fig. 3.4.2.2 Schéma de principe d'un régulateur PID. Les termes K_i et K_d sont en général exprimés sous la forme de $1/T_i$ et T_d, ce qui rend explicite leur rôle de normalisation par rapport au temps

Composante proportionnelle. L'opération la plus simple est l'amplification du signal d'erreur. Le principe en est que le signal de commande partant vers le système entraîné doit agir d'autant plus que l'on se trouve loin de l'équilibre. Avec un gain fixe, K, le signal de sortie est donc proportionnel à ε:

$$Y_P = K\,\varepsilon$$

L'avantage de la composante P c'est sa simplicité, et le fait qu'elle marche bien; elle n'est pas toujours suffisante, mais elle est toujours nécessaire.

Le problème principal provient du fait que pour avoir une action non nulle sur le système entraîné (Y>0), il faut également que l'erreur ε soit non nulle. Lorsqu'une charge constante est appliquée au système (par exemple, l'effet de la gravité sur une articulation), nécessitant un niveau de commande de V, l'équilibre n'est atteint qu'au prix d'une erreur résiduelle, appelée l'erreur statique:

$$\varepsilon = V / K$$

Bien des méthodes existent pour trouver une bonne valeur de K. Dans le principe, on choisit une valeur de K aussi grande que possible. K est en général limité soit par des phénomènes de saturation (surcharge du système, ou non-linéarités à éviter), soit par des oscillations.

Les oscillations se produisent typiquement lorsqu'une grandeur élevée de K s'accompagne de phénomènes de retard ou d'"inertie" dans le système à régler. Ceci se comprend intuitivement par le fait que tout signal alternatif change de signe après un certain retard; ainsi, une commande a priori prévue pour annuler une erreur instantanée risque, le retard aidant, d'en doubler l'amplitude...

Composante intégrale. Dans certains cas, le régulateur P peut s'améliorer par l'adjonction d'une composante intégrale.

$$Y_I(t) = \frac{1}{T_i} \int_0^t \varepsilon \cdot d\tau$$

L'idée maîtresse ici est qu'une erreur ε même petite peut, par intégration au cours du temps, conduire à une action importante. Si la sortie de l'intégrateur n'est pas envoyée au système, on la voit croître linéairement au cours du temps, dans le cas d'une erreur statique. En la branchant effectivement sur la sortie, elle parvient tôt ou tard à compenser la charge constante.

Contrôler un système par voie intégrale comprend aussi ses limites. Il y en a principalement trois.

- D'une part, le résultat de l'intégration est difficile à gérer au cours du temps. Dans une première phase, admettons que par intégration de ε on a pu générer la commande permettant de compenser une charge V. Que se passe-t-il si celle-ci disparaît? Le système va maintenant être entraîné par la composante intégrale bien au delà de la consigne. Il y restera jusqu'à ce que l'intégrale de l'erreur retombe à zéro.

- Dans un système où plusieurs régulateurs coopèrent, un seul peut avoir une composante intégrale. Si ce n'est pas le cas, une différence même infime entre leurs valeurs de consignes ou leurs valeurs de mesures conduira à l'arrêt de l'un des régulateurs, ou à la saturation d'un autre.

- Le troisième effet gênant (mais c'est en partie le premier qui resurgit sous une autre forme...), c'est qu'un intégrateur ajoute un élément de retard au système à régler. Or nous avons déjà dit que les retards conduisent souvent à des instabilités; ou tout au moins, qu'ils le ralentissent.

Composante dérivée. La composante dérivée est typiquement ajoutée à un régulateur P lorsqu'on veut en améliorer les performances (rapidité, erreur statique) sans que le système n'oscille. Le terme dérivé n'a pas d'effet direct intéressant, mais ce qui est utile, c'est qu'il permet dans une certaine mesure d'augmenter le terme proportionnel, le gain, sans dommage.

$$Y_D(t) = T_d \frac{d\varepsilon}{dt}$$

Intuitivement, on peut interpréter l'effet bénéfique de la composante D de plusieurs manières. Imaginons un système en déséquilibre (C≠M), au moment où il tend à se stabiliser. Lorsque l'on ajoute à la position, une fonction linéaire positive de la vitesse, cela permet d'anticiper un dépassement de la consigne, et donc de "freiner" assez tôt.

On peut aussi considérer qu'en réglage PD nous avons l'équivalent d'un réglage proportionnel qui serait appliqué non pas sur la mesure courante, mais sur sa prévision à l'instant $t+T_d$. Si cette prévision était parfaite, on pourrait ainsi compenser un éventuel retard de même durée, qui caractérise le système à régler. On pourrait alors sans danger d'oscillation augmenter K jusqu'à des valeurs arbitrairement grandes. Cette situation se rencontre rarement, parce que la dérivée d'un signal est un outil bien pauvre de prévision, surtout lorsque le signal à prévoir n'est pas une simple droite, qu'il émane d'un système réel, et que de plus l'horizon de prévision s'allonge.

Il faut remarquer enfin que le terme dérivé est souvent prélevé, dans les systèmes analogiques, par une génératrice tachymétrique entraînée par l'axe en mouvement. Ceci permet d'éviter une dérivation électronique du signal de position (ou indirectement, de ε), qui amplifie toujours le bruit, particulièrement aux fréquences élevées.

Le réglage d'un régulateur PID n'est pas une opération aisée puisqu'il s'agit de régler trois paramètres, à savoir : K, T_i et T_d. Ce réglage a donné lieu à de nombreux tableaux et méthodes. Dans certains cas l'approche est manuelle, expérimentale ou analytique. De plus en plus souvent aussi, ce réglage se fait de façon automatique..

Méthode de Ziegler-Nichols pour dimensionner un régulateur PID

Ziegler-Nichols ont proposé diverses méthodes pour dimensionner un régulateur PID.

Celle que nous présentons ici est particulièrement intéressante ; simple et fondamentale. C'est une méthode qui est de nature expérimentale. Il s'agit de brancher le régulateur au système avec des gains P, I et D initialement nuls, puis d'augmenter progressivement le gain de la branche proportionnelle (K_P).

Dans certains cas on peut augmenter ce gain arbitrairement et cela va de mieux en mieux. Un réglage tout-ou-rien est alors indiqué.

Dans d'autres cas, le système tend à osciller même avec un gain très faible, et il faut changer d'approche. On ne peut utiliser un régulateur PID dans ce contexte (cf. systèmes hiérarchisés).

Le cas intéressant ici est celui où un régulateur PID (ou P, PI, etc.) est à la fois nécessaire et utile.

Dans ce cas on observe généralement que l'augmentation de K_P est d'abord judicieuse (amélioration en terme de précision, de vitesse de régulation), jusqu'à une certaine valeur critique, Kc, au-delà de laquelle le système se met à osciller.

La méthode consiste essentiellement à relever la valeur du gain critique K_C, et celle de la pulsation ω_C. des oscillations qui s'installent dans ces conditions.

On choisit alors:

$K_P=0.59\ K_c$; $K_D=\pi/\omega_C$ (parfois aussi appelé T_D); $K_I=\omega_C/0.75$ (parfois aussi appelé $1/T_I$);

Ceci est évidemment une solution de principe, qui peut s'adapter, suivant l'application, aussi par exemple pour le cas d'un simple régulateur P, ou au contraire pour des cas plus évolués, par exemple avec des composantes a priori supplémentaires

B REGLAGE NUMERIQUE

Le réglage analogique a beaucoup apporté, en matière de systèmes automatiques. Néanmoins, il se fait maintenant remplacer de plus en plus par des techniques digitales (autre mot, directement dérivé de l'anglais, pour signifier "numérique"). Ces dernières sont inférieures aux précédentes sur quelques points, mais dans l'ensemble offrent surtout des avantages.

Désavantages. Nous voyons trois défauts principaux:

- Un régulateur numérique ajoute toujours une composante de retard au système à régler, la période d'échantillonnage.
- En général l'unité de calcul doit au cours du temps successivement servir à des tâches différentes, alors que les circuits analogiques fonctionnent naturellement en parallèle, chacun assumant une tâche spécifique. Les performances rapides en temps-réel sont donc beaucoup plus difficiles à assurer.
- Enfin, les techniques numériques tendent à se complexifier de façon non-linéaire et non stationnaire, ce qui pose des défis théoriques moins facilement modélisables de façon mathématique qu'en technique analogique.

Avantages. Les circuits numériques peuvent, dans une large mesure, simuler ce que font les circuits analogiques. Mais ils apportent aussi de nouvelles possibilités. Voyons d'abord les avantages qu'ils présentent lorsqu'ils se substituent simplement aux systèmes analogiques:

- Les circuits numériques n'ont pas de dérive. Un zéro, par exemple, reste un nul parfait, aussi longtemps qu'on le désire, ce qui n'est jamais le cas en technique analogique.
- Ils ont une immunité au bruit (signaux indésirables) remarquable. Tant que l'amplitude du bruit n'excède pas une certaine marge (typiquement, cette marge est grande, atteignant 40% d'un signal intéressant), son effet est nul car le signal numérique reste dans la plage de tolérance d'un état défini. Il est ainsi possible d'avoir des signaux de précision arbitrairement grande. En pratique, on est bien sûr limité par la nature analogique des signaux entrant et sortant du système à régler.
- Les circuits numériques sont maintenant moins coûteux. Cela s'explique au moins de deux façons complémentaires. D'une part, un même circuit numérique peut généralement remplacer beaucoup d'alternatives analogiques, car il est programmable. L'autre raison relève de la technologie. Les circuits numériques sont essentiellement réalisés sous forme intégrée, dans le silicium. Le coût marginal de production est incroyablement faible. Et lorsqu'ils sont produits en masse, les coûts de conception et de mise en route sont aussi, par unité, avantageux.

Les techniques numériques apportent en plus la possibilité de modifier radicalement, au cours du temps, la manière avec laquelle les entrées affectent les sorties. L'instruction fondamentale, c'est le "Si ... alors ... sinon ... ", ou, sous forme matérielle, la porte logique (N)AND. Les conséquences sont formidables, allant de l'implémentation de procédés banals tels qu'une recette de cuisine, jusqu'au fameux "distinguo" philosophique! Concrètement, on pourra ainsi facilement modifier des paramètres de régulation (tels les K, T_i et T_d) au cours du temps, mais on pourra aussi créer de nouveaux algorithmes, inimaginables en technique analogique.

Les techniques classiques en réglage analogique font une part belle aux systèmes linéaires. Dans un premier temps, elles ont été reprises telles quelles (cf par ex. : régulateur PID numérique) ou très légèrement adaptées telle la transformée en z.

Certains outils de base, tels que la transformée de Fourier, restent indispensables, mais la plupart des méthodes sont aujourd'hui dépassées, soit à cause de la nature non-linéaire des nouveaux systèmes réglés (la sortie de la somme de 2 entrées n'est pas égale à la somme des 2 sorties correspondantes...), soit à cause de leur non-stationnarité (changement de caractéristiques au cours du temps).

L'approche la plus prometteuse est de simuler, selon un maximum d'aspects, le monde réel. On est aidé dans cette démarche par la physique, dont l'objet depuis des millénaires est la représentation formelle du monde, c'est-à-dire sa modélisation.

Prenons par exemple la servocommande contrôlant le déplacement vertical d'un robot. Dans un premier temps, on y mettra un régulateur P. Si des performances supplémentaires sont désirées, un terme dérivé est utile. Il serait sage aussi d'ajouter un terme parallèle au régulateur (appelé parfois composante "a priori" ou encore "forward") compensant l'effet de la gravité. Le terme peut être constant, correspondant à la charge moyenne, ou variable, ajusté au cours du temps en fonction des variations de la charge utile. Lorsque le préhenseur est proche des points de consignes, que sa vitesse y est basse, il est tentant d'introduire une composante intégrale, afin d'annuler l'erreur statique. Mais si la consigne varie, il faut aussitôt supprimer à nouveau la composante intégrale, sinon elle va provoquer des dépassements...

La difficulté de trouver de bons régulateurs numériques s'explique en grande partie par le fait que le problème abordé est maintenant plus vaste et plus complexe qu'au temps de l'analogique exclusif. La technique a plus d'ambition. La grande entreprise européenne de l'automatisation Schneider Electric, par exemple, propose, au-delà de variantes numériques de régulateurs classiques, le recours à la logique floue et aux réseaux neuronaux.

3.4.3 REGULATION EN BOUCLE OUVERTE

Nous avons déjà traité de régulation en boucle ouverte à un niveau hiérarchique supérieur. Dans la partie 3.3, le problème de la coordination des articulations est étudié de façon détaillée, et dans ce contexte-là, les trajectoires sont presque toujours calculées en boucle ouverte. On attend des servocommandes en aval, qu'elles entraînent exactement les articulations selon les consignes successives transmises. Néanmoins, pour qu'elles puissent effectivement suivre, on respecte des lois de mouvements plus ou moins douces.

Au niveau le plus bas des commandes de moteurs, le problème est en bien des points similaire.

Tous les moteurs électriques ne peuvent pas se commander en boucle ouverte. Les moteurs à courant continu par exemple ont une courbe théorique linéaire, à charge constante, qui lie vitesse et tension d'alimentation. Mais en pratique la charge n'est jamais constante, ni même connue, aussi ne peut-on prévoir effectivement la vitesse ou encore plus la position du moteur sur la seule base de la commande.

Ce sont les moteurs pas-à-pas que l'on commande en boucle ouverte. Les bobines du moteur sont alimentés par phases (moteurs à 2, 4 ... 8 phases), à l'aide de circuits électroniques comprenant un séquenceur. Le séquenceur gère la succession des phases alimentées, et à chaque impulsion reçue de la commande, le moteur avance ou recule d'un pas.

En pratique, c'est la loi de mouvement à profil de vitesse trapézoïdal qui est le plus souvent employée pour assurer des mouvements doux et fiables. Démarrage et arrêt se font à accélération constante, alors que le gros du mouvement se fait à vitesse constante (voir § 3.3.2.B).

3.4.4 EXEMPLE DE CIRCUIT DE COMMANDE POUR MOTEUR: LE HCTL-1000

Dans les années 80, Hewlet-Packard avait étudié la possibilité de s'engager dans le domaine de la robotique. Parmi les retombées de leur premiers essais, il se trouve des capteurs optoélectroniques incrémentaux, ainsi que le circuit numérique spécialisé pour la commande de moteurs, le HCTL-1000. Le circuit a toujours un succès considérable aujourd'hui, et mérite que l'on s'y arrête. Bien qu'une tendance nouvelle bien différente apparaît également, qui se base sur l'utilisation de tout un PC sur un hardware très compact et avec un logiciel multi-tâche temps résel compatible avec Windows (par exemple : VxWorks).

Le circuit se programme par un microprocesseur ou un ordinateur externe à travers un canal dont le débit peut être faible. En aval, il commande des circuits d'amplification, puis le moteur qui lui-même entraîne la charge. Le circuit peut commander le système en boucle ouverte (cas des moteurs pas-à-pas) ou en boucle fermée (moteurs à courant continu, ou synchrones autocommutés). Dans le second cas, un capteur incrémental renvoit vers le circuit une information de position.

La fig. 3.4.3 présente de façon schématique les nombreuses fonctions assurées par le circuit. Voyons en les principales.

Gestion du capteur incrémental de position. Le bloc "Quadrature decoder / counter" permet d'interpréter les signaux provenant d'un capteur incrémental (voir § 1.2) afin de détecter le sens de rotation de l'axe, ainsi que de mettre à jour d'un compteur de position dont la capacité est de 24 bit.

Calcul d'erreur. On se souvient que la première opération d'un régulateur consiste à évaluer l'"erreur", c'est-à-dire la différence entre consigne et mesure. Pour cela, le HCTL-1000 utilise trois blocs: "∑", "Input command", et "feedback". Suivant les cas, le circuit peut travailler sur des informations de position ou de vitesse.

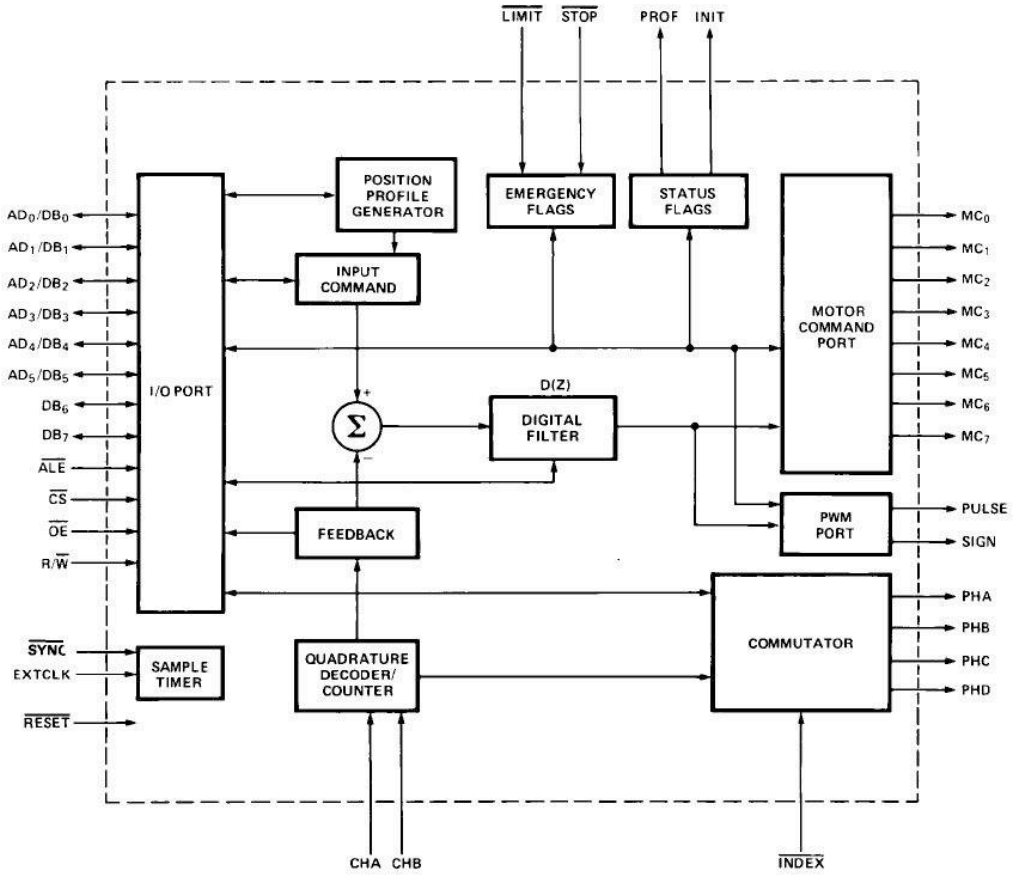


Fig. 3.4.3.1 Représentation schématique des fonctions du circuit de régulation HCTL-1000.

Composantes PID. Le circuit implémente un régulateur PID en technique numérique (bloc "Digital filter"). Dans le jargon approprié, nous travaillons avec la transformée en z. Le régulateur est décrit par sa fonction de transfert, ce qui le rend équivalent à un filtre. La composante différentielle, caractérisée en technique analogique par T_d , se traduit ici par un "zéro", au numérateur, alors que le paramètre intégral, T_i, prend l'aspect d'un pôle, au dénominateur. Les bonnes valeurs du régulateur peuvent se calculer selon des méthodes essentiellement similaires à celles qui sont utilisées pour les systèmes analogiques. Souvent, les valeurs sont aussi choisies par tâtonnement... Rappelons que le problème se complique souvent parce que le système n'est pas stationnaire. En particulier, la charge varie souvent beaucoup, au cours du temps.

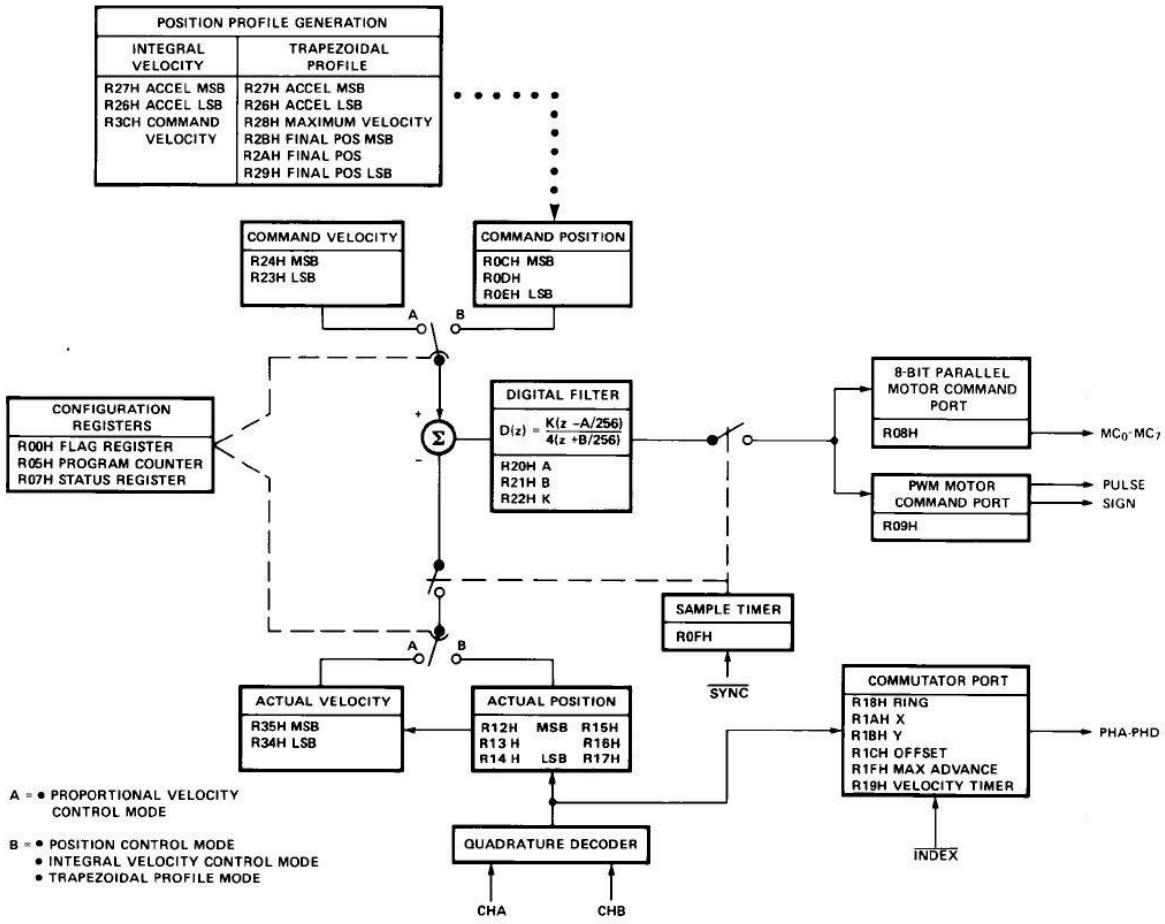


Fig. 3.4.3.2 Représentation plus claire des fonctions du circuit de régulation HCTL-1000/1100.

Interpolation. Le bloc "Position profile generator" a pour fonction de générer un déplacement entre consignes fixées par l'utilisateur, selon la loi de mouvement à profil trapézoïdal de vitesse (voir § 3.3.2.B). Il exploite, en plus des consignes de position, des ordres de vitesse et d'accélération maximales.

Circuits de sortie. Le HCTL-1000 contient vraiment une palette complète de modes de commande de moteur. Il peut gérer aussi bien un moteur à courant continu (CC) qu'un moteur synchrone autopiloté (MSA), ou qu'un moteur pas-à-pas (PAP). De plus, dans les deux premiers cas, la consigne de tension destinée aux amplificateurs peut à choix être linéaire, ou binaire, codant les niveaux d'amplitude différents par modulation de largeur d'impulsions. Les blocs mis en jeu sont décrits un par un.

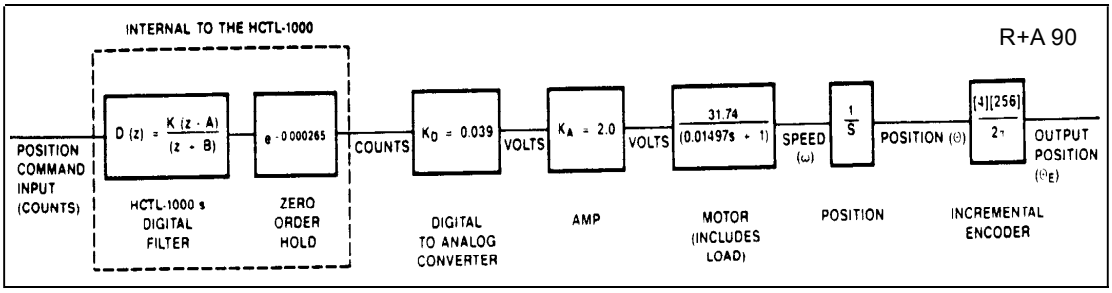


Fig. 3.4.4 Modèle du circuit régulateur, ainsi que de l'ensemble du système en boucle ouverte. De tels modèles étaient présentés comme utiles, il y a 25 ans, pour un choix systématique ("optimal") des paramètres de régulation.

Gestion des enroulements de moteur. Le bloc "Commutator" gère une séquence d'états. Par pas successifs, différents enroulements, ou groupe d'enroulements ("phases") d'un moteur pas-à-pas, ou synchrone autopiloté, sont alimentées. Dans le cas du PAP, c'est l'utilisateur qui fixe la succession de pas désirés, de façon semblable à ce qui a été décrit ci-dessus (§ *Interpolation*). Pour le MSA, ce sont les signaux du capteur incrémental qui commandent l'avance de pas. Le circuit peut commander directement jusqu'à 4 phases. Au-delà (m phases), il faut interpréter ces 4 sorties comme l'adresse d'une table à 16 mots et m sorties.

Plusieurs raffinements sont possibles: gestion de capteurs de position à nombre d'incréments différents du nombre de pas du moteur; "avance à l'allumage" des phases, afin de compenser les constantes de temps de l'amplificateur et des bobines, sans avoir recours à des réseaux extérieurs de résistances et condensateurs; superposition de phases successives, etc.

Sortie linéaire. Le bloc "Motor command port" fournit 8 sorties binaires parallèles, destinées à un convertisseur numérique/analogique externe. C'est donc pratiquement un signal linéaire que l'on obtient, utilisable soit en tension, soit en courant, pour, à travers un ampli, entraîner un moteur. Il s'agit de façon indifférente ici d'un moteur CC ou MSA. Notons dans ce dernier cas, que le circuit dispose donc de 12 lignes pour la commande du moteur (utilisation simultanée de la sortie linéaire et de la commande de phases).

Avec 8 bit en sortie, la commande linéaire peut paraître peu précise. Ce n'est pas du tout le cas. Ce qui fixe la précision d'un mouvement, ce n'est pas simplement le nombre de niveaux possibles de tension aux bornes du moteur (dans certains cas, celui-ci vaut 2...) mais c'est surtout la résolution du capteur de position. La position, pour un robot industriel, est généralement codée sur un minimum de 16 bit. Le présent circuit en gère 24!

Modulation de largeur d'impulsions. Le dernier bloc discuté ici, c'est le "PWM Port". Comme alternative à la sortie "linéaire", qui nécessite un convertisseur externe, le circuit offre une sortie binaire modulée dans le temps. Cette sortie peut directement attaquer certains amplificateurs.

A l'intérieur d'une période de durée fixe, T, une faible vitesse du moteur sera codée par une impulsion brève, alors qu'au contraire un signal précédemment décrit comme élevé apparaîtra maintenant comme une impulsion longue, couvrant quasiment toute la période T. Durant l'impulsion, on peut considérer que le moteur est alimenté, alors qu'il est "en l'air" le reste du temps.

Une deuxième sortie de 1 bit fixe la direction du mouvement.

Le circuit HCTL-1000 est un circuit performant, qui allège l'hôte dont il dépend de toutes les opérations rapides intervenant dans la commande de moteurs. Mais il a aussi ses limites, qui sont de deux types. D'une part, il n'est pas assez rapide encore pour certaines applications de pointes. D'autre part, certains lui préfère les micro-contrôleurs "classiques".

Les micro-contrôleurs sont très "compacts". Ce sont des microordinateurs monopuces, où des convertisseurs analogiques ainsi que des ports séries de communication sont disponibles. De plus, leur unité centrale de traitement est souvent compatible avec des systèmes de développement plus grands, où l'interface entre machine et utilisateur ainsi que l'environnement de programmation sont tous deux efficaces.

3.5 QUELQUES LANGAGES POUR LA ROBOTIQUE ET L'AUTOMATISATION

Alors que dans la partie 3.2, le problème général de programmation d'une cellule de fabrication a été traité, c'est ici l'étude concrète de divers langages utilisés dans l'industrie que nous allons aborder.

Les langages retenus sont principalement les suivants: Rapid/ABB, VAL+/Stäubli, ARIA, et Piaget. Le choix a été guidé par deux types de considérations. D'une part, les principaux langages existants doivent être représentés. D'autre part, pour une classe donnée de programmes, la priorité va au langage disponible au laboratoire de robotique et d'automatisation de l'EIVD.

Chaque langage fait d'abord l'objet d'une brève description individuelle. Ensuite le lecteur trouvera un tableau comparatif rassemblant leurs caractéristiques majeures. Pour des informations plus détaillées sur les langages et les systèmes sur lesquels ils tournent, on se reportera à des fascicules annexes (3.5.A: ABB, 3.5.C: VAL-2, 3.5.E: ARIA, 3.5.F: Comau), ou pour plus d'informations encore, à la documentation technique du fabricant, disponible au laboratoire.

3.5.1 RAPID/ABB

Le langage ABB (Asea Brown Boveri) est important vu l'impact majeur de ces robots au niveau mondial. De façon surprenante au premier abord, l'interface utilisateur en limite de façon drastique la programmabilité: pas de clavier ni de console au sens courant. La communication se fait par un boîtier portable doté d'une vingtaine de boutons et de deux lignes de texte.

Le langage ABB est tout-à-fait représentatif d'autres langages pour robots et pour l'automatisation qui se caractérisent par une approche typique des commandes de machines-outils traditionnelles. C'est en particulier le cas du langage SRL de Siemens (utilisées par exemple sur divers robots allemands tels Kuka ou Manutec).

Alors que la version standard se limite aux fonctions de base, minimales, toutes sortes d'extensions ont été montrées par le fabricant sur des sites pilotes: vision intégrée, trajectoires asservies à des capteurs de force, programmation hors-site en contexte CAO, etc.

3.5.2 VAL (V+)

VAL est un langage puissant pour robots industriels dérivé de AL. AL a été développé à l'Université de Stanford, et reste une référence classique.

VAL a été développé pour les robots Unimation, et s'utilise actuellement aussi pour les robots Stäubli et Adept . Dans les versions récentes, des extensions importantes ont été introduites pour la programmation structurée, les trajectoires adaptatives et pour la communication en réseau.

V+ est aussi représentatif d'autres langages, similaires, optimisés pour la gestion de référentiels multiples. Le langage Karel des robots Fanuc (ex-GMF), Renault, et Comau ("KL2"), en sont des exemples.

3.5.3 ARIA

ARIA est un langage récent pour robots industriels développé par Demaurex Robotique et Microtechnique. Ce langage est basé sur Pascal, et à ce titre, il est représentatif de plusieurs autres langages pour robots, tel RAIL aux Etats-Unis.

Mais il a en plus diverses particularités intéressantes: travail à très haute vitesse, trajectoires interpolées de façon linéaire ou elliptique, programmation en réseau, avec fonctionnalités réparties, intégration de systèmes d'analyse d'images.

Plus qu'un simple langage, ARIA constitue en fait un environnement de travail à plusieurs couches, qui permet de commander des systèmes divers, allant d'un robot isolé à toute une ligne de production.

3.5.4 PIAGET

Piaget est un langage multi-agents (multi-tâches), temps réel, créé à la HEIG-VD dans le but de programmer rapidement des robots, mobiles autonomes ou industriels.

Dans sa première version (1997-2001), Piaget était très spécialisé, avec des instructions du genre ChoisirLePontVisuellement, ActionnerFusée, ou encore TirerSurTour(centrale).

Puis dès fin 2001, on est revenu un peu vers des instructions plus générales, et le bon compromis s'est établi au niveau des instructions spécialisées pour robots. Stratégiquement, des instructions du type de V+ ont été introduites pour les robots mobiles autonomes (Move, Appro, Signal, etc.), comme base de programmation..

3.5.5 AUTRES LANGAGES

Parmi les autres langages pour robots, on peut notamment citer, également représentés au laboratoire de robotique et automatisation (LaRA), le langage KRL de Kuka, très inspiré de Val et de Rapid, et l'environnemnt Chrégraphe pour la programmation de l'humanoïde NAO.

3.5.6 TABLEAU COMPARATIF

Les paragraphes précédant ne présentent, pour chaque langage mentionné, que quelques éléments saillants. On se propose ici de décrire leurs instructions de base : lecture des entrées, changement de l'état dess sorties, lecture de la position du bras et commande de mouvement.

Quel système est le meilleur ? Il n'est pas possible de répondre en toute généralité à cette question. Si un paramètre est critique pour une prise de décision, il conviendra d'évaluer plus à fond ce que tel ou tel système apporte. En fin de compte, un benchmark très proche de l'application envisagée reste le meilleur critère d'appréciation. Pour cela, la disponibilité de la plupart de ces systèmes au laboratoire s'avère extrêmement précieuse.

Récapitulatif des langages pour robots disponibles au laboratoire C38-exC01 (1 de 7)

Langage Instruction	Piaget HEIG-VD.iAi.LaRA	V+ Stäubli/Adept	Val-3 Stäubli
Lire une entrée	SignalIn(No)	SIG(1000+No)	dio Bouton // *déclarations* dioLink(Bouton,io:bIn0) ex: if (Bouton) ... // *utilisation*
Imposer une sortie	SignalOutAGN(No, true ou false)	SIGNAL No ou -No	// *Exemple: variable de sortie: LED* dio LED // *déclarations* dioLink(LED,io:bOut0) dioSet (LED,true) // *utilisation*
Demander un mouvement	MoveAGN(location)	Move ou MoveS location	movej(pPremierPoint,flange,mNomSpeed) movel ...
Attendre fin de mouvement	implicite	Break	WaitEndMove()
Lire une position	Here() ou P0Robot	HERE	here(flange,world)
Définition des positions par apprentissage	néant	Boutons poussoirs, avec coordonnées articulaires; ou spatiales, en repères de base ou d'outil	idem

Définition explicite	Trans(x,y,phi)
	P.x= .., P.y=....
		Trans(x,y,z,phi, theta,psi); MTrans(m_0...); rot Euler ZYZ	Ex. trTransIZ100={0,0,100,0,0,0}
			point p; p = {{100, -50, 200, 0, 0, 0}, {sfree, ofree, wfree}} // x,y,z,rx,ry,rz,
			p.trsf.x=100 // mm, rot Euler XYZ

Récapitulatif des langages pour robots disponibles au laboratoire C38-exC01 (2 de 7)

Langage Instruction	Gemini Bosch-Delta	ARIA SIG-Pack Demaurex-Delta
Lire une entrée	Input R1_Switch1 Read	Ax_Q_Input(NoCarte, No)
Imposer une sortie	Output R1_SUCTION1 Write 1 *(ou R1_BLOWING1)*	Ax_Output (NoCarte, No, 0 ou 1)
Demander un mouvement	Robot R1 MoveToPoint %PositionX %PositionY %PositionZ Robot R1 RelativeMoveTo [X] [Y] [Z] [A]	D3n_Absmove(1,x,y,z) Ax_ Absmove (NoCarte, theta)
Attendre fin de mouvement	TimeWait 500 *% en millisecondes* *%: commentaire ou préfixe de variables*	D3n_Q_EndMoveReached(1) Ax_Q_EndMoveReached (NoCarte)
Lire une position	%PositionX, %PositionY, %PositionZ Robot R1 GetPosition	D3n_Q_Target(1,x,y,z) Ax_Q_Target(NoCarte, theta)
Définition des positions par apprentissage	Coupure de puissance dans le bras ; déplacement du bras à la main ; et lecture à l'écran de la position en coordonnées cartésiennes ou articulaires	Coupure de puissance dans le bras ; déplacement du bras à la main ; et lecture à l'écran de la position en coordonnées cartésiennes ou articulaires
Définition explicite	Var PositionX Write 280	Cf. D3n_Absmove(1,x,y,z)

Cf.GeminI_4-Instructions&ErrorsManual 015.10.25.pdf

Voir aussi solution en C# de HEIG-IAI-LaRA

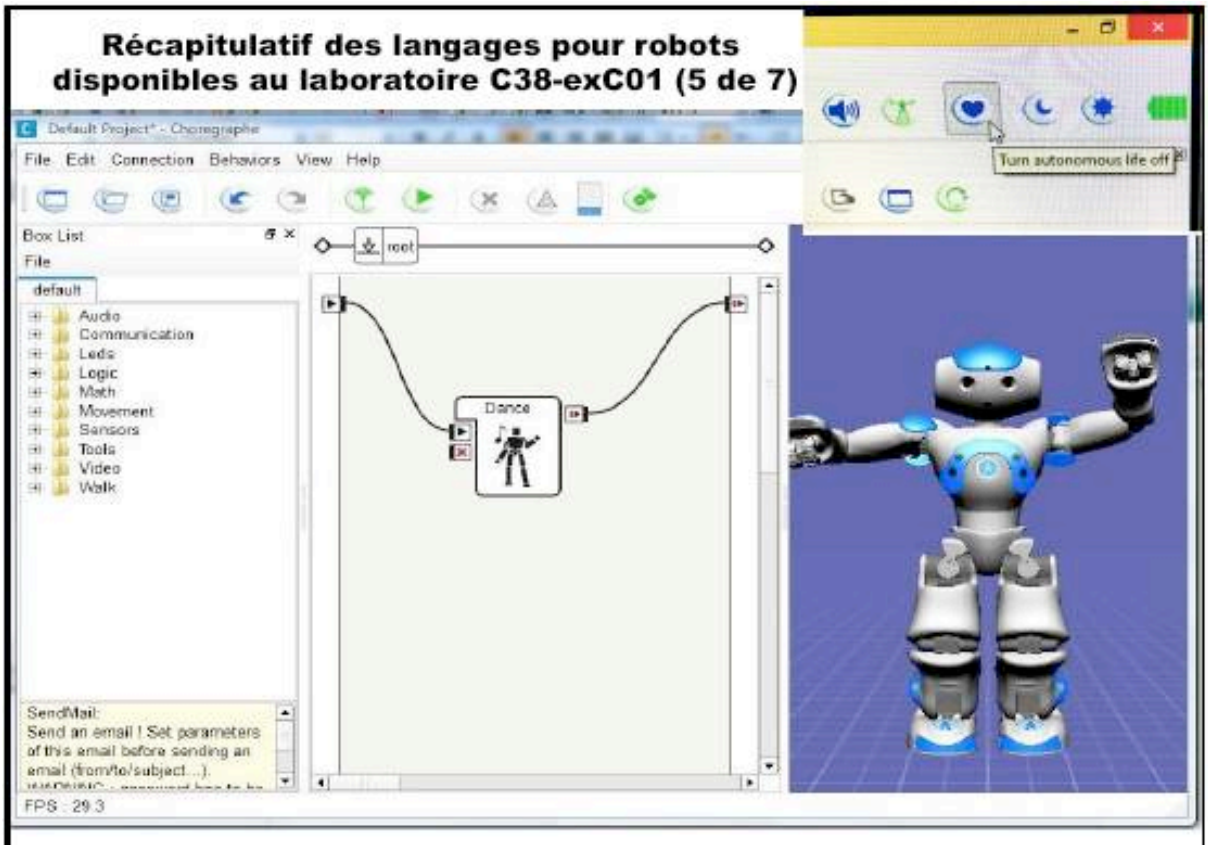
Récapitulatif des langages pour robots disponibles au laboratoire C38-exC01 (3 de 7)

Langage Instruction	Rapid ABB	EZ-PDL2 Comau	KRL Kuka
Lire une entrée	diNo	$DIN[No]	Wait, $IN[n]
Imposer une sortie	Set doNo, high ou low Set ou Reset doNo	$DOUT[No]=ON ou OFF	$OUT[n]=true
Demander un mouvement	MoveJ ou MoveL RobTarget , v500, fine, tool1	Move Linear ou Joint To Position	PTP P1 Ou LIN P2 PTP_REL, LIN_REL, CIRC_REL
Attendre fin de mouvement	Wait /InPos		
Lire une position	CrobT CJointT	ARM_POS ou ARM_JNTP	
Définition des positions par apprentissage	Joystick, avec coordonnées articulaires; ou spatiales, en repères de base ou d'outil	Boutons poussoirs, avec coord.articulaires; ou spatiales, en repères de base ou d'outil	idem
Définition explicite	p.trans.x=..;p.trans.y=..; ..; p.rot.q1=..; ..; p.rot= OrientZYX(Rz,Ry,Rx);	p.x=..; .. ; p.z=..; p.r=..; ..; p.w= ..; X,Y,Z,EulerZYZ	Cible.x=.., Cible.y=..

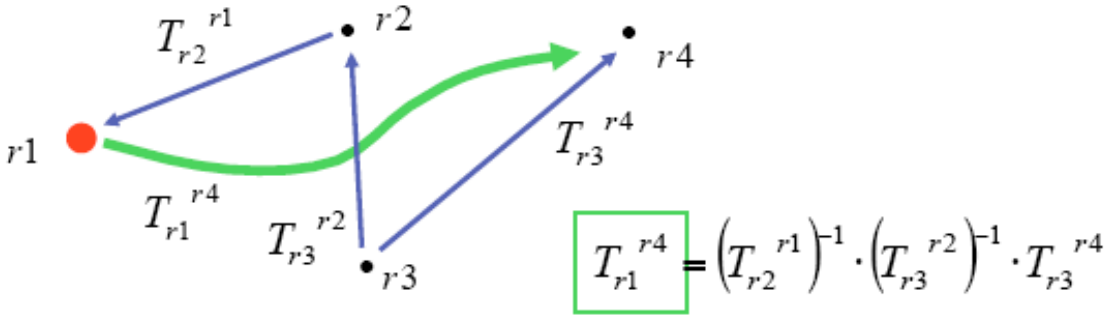
Récapitulatif des langages pour robots disponibles au laboratoire C38-exC01 (4 de 7)

Langage Instruction	Katana	Katana - Cartesian
Lire une entrée	• int IO_ReadInput (int inputNr) reads an input from the digital I/O	
Imposer une sortie	int IO_SetOutput (int outputNr, int value) sets an output of the digital I/Os	closeGripper();
Demander un mouvement	int MoveMot (int axis, int enc) PTP movement. int MoveToPosEnc (int enc1, int enc2, int enc3, int enc4, int enc5, int enc6, int velocity, int acceleration, int tolerance) Moves all axes to a target encoder value. .	moveToPos(tmpPos, DEFAULT_SPEED, DEFAULT_ACCELERATION); moveToPosLin(tmpPos, DEFAULT_SPEED, DEFAULT_ACCELERATION);
Attendre fin de mouvement	int WaitForMot (int axis, int targetpos, int tolerance, int mode) waits until the axis is back in hold state	
Lire une position	int getState (int axis) gets the axis state • int getEncoder (int axis) gets the position	getPosition(tmpPos);
Définition des positions par apprentissage	Teaching en coupant la puissance et bougeant le bras	Teaching en coupant la puissance et bougeant le bras
Définition explicite		tmpPos->X, etc Y,Z, phi...,EulerZYZ

Récapitulatif des langages pour robots disponibles au laboratoire C38-exC01 (5 de 7)



Récapitulatif des langages pour robots disponibles au laboratoire C38-exC01 (6 de 7)



Fonctions disponibles pour évaluer ce type de graphe ou d'équation matricielle dans des langages de robot:

	Matrice inverse	*Produit matriciel*
Stäubli/V+:	**set tinv=inverse(t)**	**t3=t1:t2**
Stäubli/Val3:	**tinv= ! t**	**t3=t1*t2**
ABB/Rapid:	**tinv:=PoseInv(t)**	**t3:=PoseMult(t1, t2)**
Piaget:	**tinv=MatriceInverse(t)**	**MultMatrices(t1,t2,t3)**

Récapitulatif des langages pour robots disponibles au laboratoire C38-exC01 (7 de 7)

Fonctions accessoires éventuellement utiles, en langage pour robot Stäubli/Val3 (cf. notions "parasites" liées aux points, repères, outils…).

Apprentissage d'un point (Teaching): pMonPoint=here(tOutil,world)

Demande de mouvement:

movej(pMonPoint, tflange, mNomSpeed)

Définition d'un point à l'origine de World: pNull=pMonPoint, puis

trNull={0,0,0,0,0,0}, puis pNull.trsf=trNull

Extraction de la partie géométrique ("transformation") d'un "point":

trMaTransformation=position(pMonPoint,fWorld)

Création d'un point à partir d'une transformation

pMonPoint=pNull, puis pMonPoint.trsf=trMaTransformation

Création d'un repère ("frame"):

nErreur=SetFrame(pOrigine, pAxeX, pPlanYX, fPalette)

Mouvement simple, relatif, à interpréter dans un repère quelconque, à définir:

pMonNouveauPoint=compose(pMonPoint, fMonRepereOuSimple, trMaTransformationSimple)