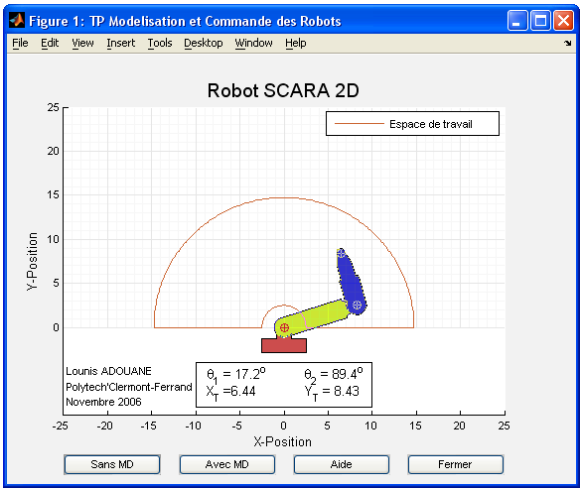
POLYTECH
CLERMONT-FERRAND

Modélisation et Commande des Robots

AURO11
Travaux Pratiques



GE3 2009-2010
Lounis ADOUANE

Organisation des séances de TP Robotique GE3

PREAMBULE

L'objectif de ces trois séances de TP d'AURO11 est de fixer, par la pratique, les principales notions de modélisation et de commande des robots manipulateurs traités en AURO10. Ceci passe inévitablement par l'étude des modèles géométriques, cinématiques et dynamiques des robots manipulateurs ainsi que par les méthodes de génération de trajectoires. D'autres notions vont par ailleurs être traitées comme celle de l'évitement d'obstacles ou de la commande par PID d'un bras manipulateur, et cela dans l'objectif de donner libre cours aux réflexions et aux initiatives propres à chaque binôme.

VUE GENERALE SUR LES TP'S

Ces séances de TP se proposent en premier lieu d'implémenter sous MATLAB les différentes modélisations élémentaires liées en grande partie à un robot SCARA plan (RR), et par la suite d'adjoindre l'ensemble de ces programmes à d'autres programmes préalablement fournis (cf. Partie I, section II) pour permettre de mettre en place un mini-simulateur visant à commander d'une manière conviviale et interactive le robot étudié.

Il est à noter par ailleurs que la simulation du robot étudié s'effectuera au début (Partie I) sans faire intervenir le modèle dynamique du robot, ce n'est que par la suite (Partie II) que le modèle dynamique sera introduit afin de traiter des problématiques de commande des robots manipulateurs.

TRAVAIL A RENDRE

Après la fin des trois séances de TP vous devez rendre un compte-rendu sous **format papier** (1 par binôme). Le contenu et la forme de ces comptes rendus seront pris en compte pour le calcul de la note finale des travaux pratiques. Ces documents doivent absolument être déposés dans ma boîte aux lettres *une semaine après la troisième séance de TP*. Vous devez également me fournir les principaux programmes (.m), (.mdl), etc. que vous allez réaliser (à m'envoyer à mon adresse email[1] avec comme objet : TP MATLAB [Nom1, Nom2].

NB : Mettez l'ensemble de vos programmes dans un seul répertoire que vous allez compresser avant de l'envoyer par email.

NOTATION

La note finale des travaux pratiques est composée de deux notes :

1. « Oral » des séances de TP : l'encadrant apprécie le comportement et l'implication de l'étudiant durant chaque séance de TP. (coef. 1/3)
2. « Écrit » note issue du compte rendu. (coef. 2/3)

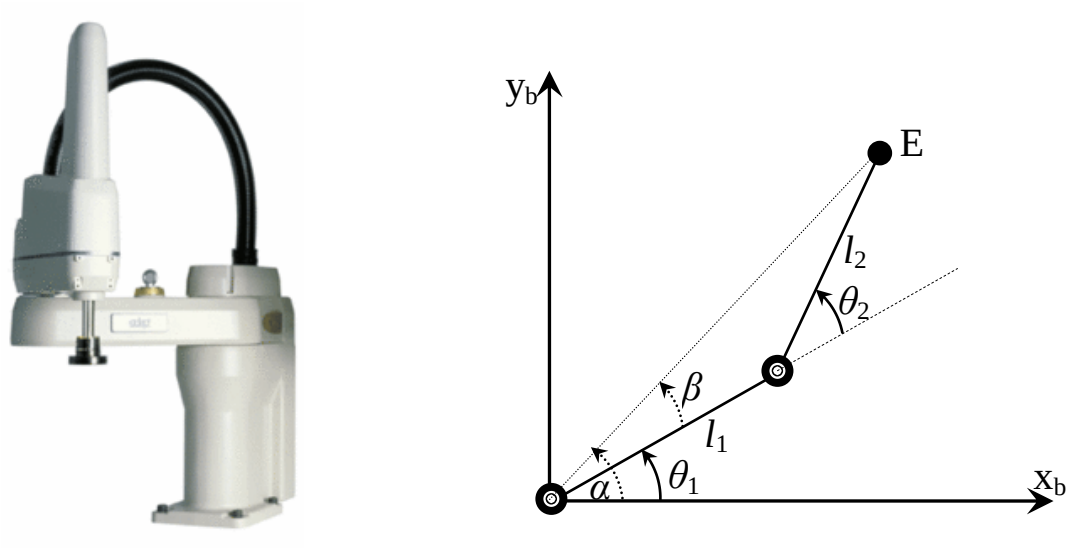
[1] Lounis.Adouane@lasmea.univ-bpclermont.fr

Partie I

Modélisation et génération de trajectoires

I Introduction

Le robot SCARA (Selective Compliance Assembly Robot Arm) (Figure.1(a)) est l'un des robots les plus utilisés en industrie. La version à deux degrés de liberté (2ddl) du SCARA va nous servir dans ce TP comme base d'étude pour la modélisation et le contrôle d'un système robotique. La configuration du robot est déterminée par les variables articulaires θ_1 et θ_2 (Figure.1(b)).



(a) **(b)**

Figure. 1

II PROJET DE ROBOTIQUE SOUS MATLAB

L'objectif de ce TP est l'utilisation de MATLAB pour modéliser et commander un robot de type SCARA à deux degrés de liberté. A ces fins, un ensemble de programmes vous sont préalablement fournis, et vous permettront de disposer d'une ossature de programmes nécessaire pour mener à bien vos TP's. D'autre part, avant d'effectuer le moindre travail, récupérez et copiez sur votre poste de travail le répertoire contenant l'ensemble des programmes sur lesquels vous devez travailler. Les fichiers et leur contenu sont les suivants :

SimulationRobotSCARA.m	C'est le programme principal, il permet en l'exécutant d'ouvrir une interface utilisateur (Figure. 2) qui vous permettra de simuler le système robotique avec ou sans l'utilisation de Simulink. Une représentation graphique du robot SCARA 2ddl ainsi que la possibilité de l'animer sont accessibles via cette interface. Cette interface est aussi disposée pour récupérer les événements émanant de la souris. En l'occurrence, la souris va vous permettre dans le cadre de votre projet de TP de donner des consignes de position (x, y) à atteindre par l'organe terminal du robot
ModeleGeometriqueDirect.m	Définit les positions (x_i, y_i) des différentes articulations du SCARA en fonction de ses coordonnées articulaires
ModeleGeometriqueInverse.m	Définit les coordonnées articulaires du SCARA en fonction de la position de son effecteur
TrajectoireConsigne.m	Donne dans le domaine opérationnel la trajectoire consigne à faire suivre par l'effecteur du SCARA
GestionGUI.m	Gère tous les événements (clic de souris, bouton enfoncé, etc.) affectant la fenêtre GUI (Graphical User Interface)
SetAffichage.m	Permet de remettre à jour la représentation graphique du robot SCARA ainsi que l'affichage des informations articulaires et la position de l'effecteur (Figure. 2)
SimulinkControleRobotSCARA_SansMD.mdl	Modèle Simulink (Figure. 3) qui permet en l'interfaçant avec les programmes décrits ci-dessus de commander le mouvement du SCARA afin de suivre une trajectoire consigne par exemple.

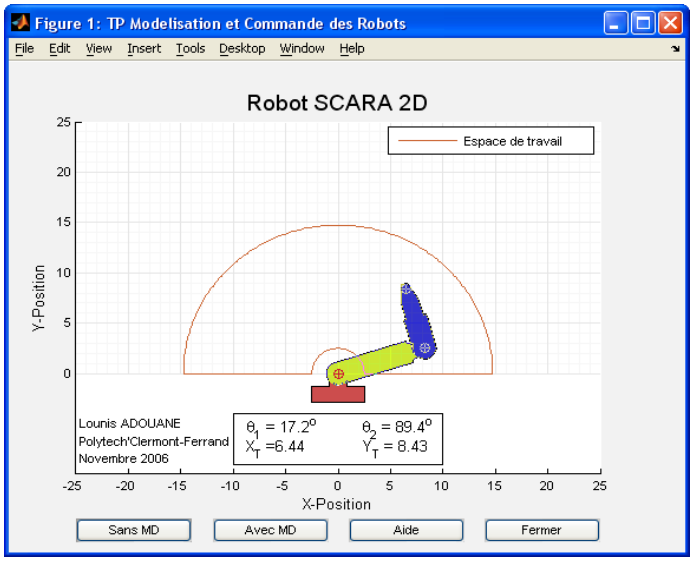


Figure. 2

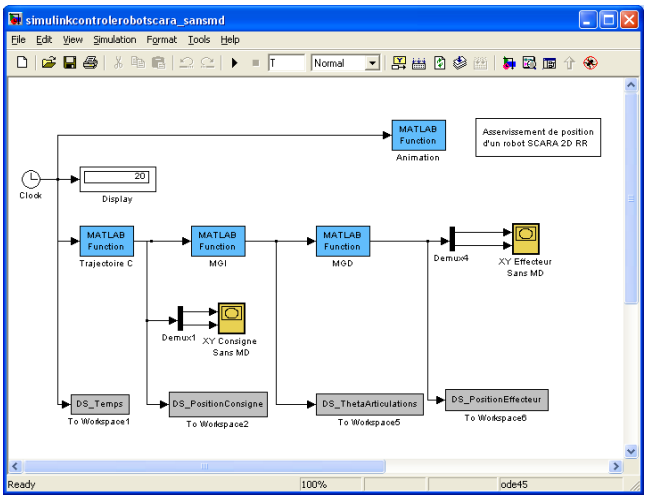


Figure. 3

III LE TRAVAIL A FAIRE

Tester et explorer **attentivement** les programmes fournis ainsi que le modèle Simulink afin de comprendre les méthodes adoptées pour réaliser les éléments préalablement accessibles pour la simulation (déterminez le rôle de la variable « T »).
Dans certains cas vous devez ajouter à certaines de ces programmes des parties de codes (pour les cas des m-files) ou des schémas-blocs (pour le cas des modèles Simulink) qui vont permettre soit d'améliorer les fonctionnalités du mini-logiciel fourni, soit d'ajouter des fonctionnalités complètement nouvelles. Il est à noter que, pour une meilleure lisibilité de vos programmes, il est **fortement conseillé de les commenter**.

III.1 Modélisation géométrique

A) Le Modèle Géométrique Direct (MGD)

A.1) En utilisant l'équation (65) du polycopié de cours d'AURO10, écrivez une fonction MATLAB qui calcule le MGD d'un robot quelconque en chaîne ouverte simple.
Indication : La fonction sera appelée par *T0Tn = MGD_General(alpha, d, theta, r)*, avec (alpha, d, theta, r) sont les vecteurs des paramètres géométriques du robot (chaque vecteur correspond à une colonne du tableau de Denavit-Hertenberg modifié).

A.2) Pour des valeurs aléatoires du vecteur $[\theta_1 \ \theta_2]^T$ vérifiez que les fonctions *MGD_General.m* et *ModeleGeometriqueDirect.m* (fonction fournie au préalable) donnent les mêmes résultats.
Indication : les longueurs spécifiques du robot étudié sont l_1=8.625 et l_2=6.125 (Figure.1b).

B) Le Modèle géométrique Inverse (MGI)

Pour le cas de structures robotiques simples telles que celle du robot SCARA plan, il est possible de trouver le MGI par un raisonnement purement géométrique. C'est ce qui est d'ailleurs implémenté au sein de la fonction *ModeleGeometriqueInverse.m* (cf. programmes fournis).

B.1) Il est demandé dans ce qui suit d'utiliser les méthodes systématiques étudiées durant le cours d'AURO10 pour obtenir le MGI et de l'implémenter dans un programme MATLAB :
La fonction doit être appelée par : *[Q1, Q2, err] = ModeleGeometriqueInverse_Bis(X)*, avec Q1 la solution avec $\theta_2 > 0$ (coude bas), Q2 la deuxième solution, err(1)=0 si la première solution est possible et =1 si la solution n'est pas possible, de même err(2) indique la faisabilité de la deuxième solution.
Indications :

- au début de la fonction testez si le point appartient à l'espace de travail du robot (cf. Fig.2),
- lorsque vous avez une racine carrée à calculer, vérifiez que l'argument est positif, sinon pas de solution,
- lorsque vous avez une expression égale à sinus ou cosinus d'un angle, il faut que la valeur absolue de cette expression soit ≤ 1, sinon le point X n'est pas accessible,
- en écrivant un MGI, on doit tester si les valeurs obtenues se trouvent dans le domaine articulaire du robot ($0° \leq \theta_1 \leq 180°$; $-170° < \theta_2 < 170°$).

B.2) Testez la fonction développée en donnant quelques valeurs pour X puis appelez le MGD avec les valeurs obtenues de Q1 ou de Q2 pour vérifier si les solutions sont les mêmes. Par ailleurs, vérifiez que les fonctions *ModeleGeometriqueInverse* et *ModeleGeometriqueInverse_Bis* donnent les mêmes résultats.

III.2 Modélisation cinématique

A.1) Programmez le Modèle Cinématique Direct et Inverse (MCD-MCI) du robot SCARA.

A.2) En interfaçant votre MCI au programme Simulink fourni (cf. Figure. 4), simulez un déplacement rectiligne et à vitesse constante de l'effecteur selon l'axe x (utilisez l'interface fournie pour observez le résultat).

A.3) Il est impératif de correctement gérer les singularités afin de prévenir tout mouvement erratique du robot. Quelles sont les positions singulières du robot étudié ? Utilisez cette connaissance pour faire en sorte que le robot ne puisse pas passer par ces configurations singulières. Commentez.
Indications :

- le seul programme à modifier est celui où vous avez défini le MCI du robot,
- pour les singularités en limite de l'espace de travail, on peut imposer des butées logicielles aux angles des axes de manière à s'arrêter juste avant la configuration « bras complètement tendu » ou « bras complètement plié ».

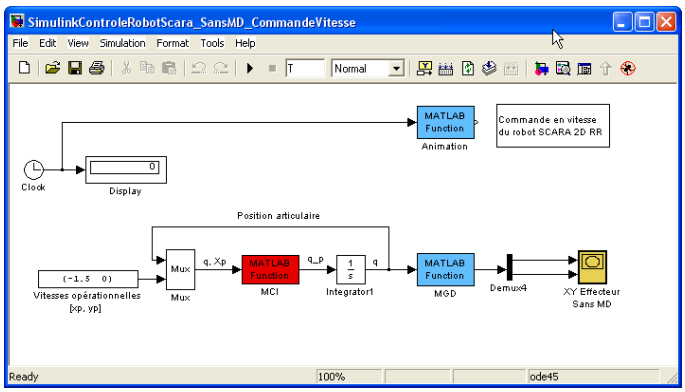


Figure. 4

III.3 Génération de trajectoires

A) Espace articulaire

A.1) Ecrivez un générateur de mouvement $(q,\ \dot{q},\ \ddot{q})$ pour le robot SCARA entre 2 points de l'espace articulaire $q^i = [100°\ 100°]^T$ et $q^f = [6°\ 60°]^T$ en utilisant un interpolateur de degré 1 et 5. Les vitesses articulaires max et accélérations max sont données par : **kv** = $[50\ \ 40]^T$.°/s et **ka** = $[100\ \ 60]^T$.°/s^2.

A.2) Dessinez l'évolution de $(q, \dot{q}, \ddot{q})$ en fonction du temps ainsi que la position de l'effecteur en fonction du temps. Utilisez les programmes fournis ainsi que ceux que vous avez implémentés pour visualiser l'évolution de vos différentes variables articulaires ainsi que celle de l'effecteur.

Indications :

- une fois le temps « Max{t_{fj}} » calculé pour chaque type d'interpolation, vous devez l'affecter à T (temps de la simulation),
- pour l'affichage de $(q, \dot{q}, \ddot{q})$ vous pouvez utiliser les fonctions MATLAB suivantes : *subplot*, *xlabe*, *ylabe*, *legend*, *hold on*,
- utilisez la fonction MENU de MATLAB pour choisir d'une manière interactive la trajectoire consigne à donner au robot.

A.3) Modifiez les générateurs de mouvement que vous avez précédemment développés afin de permettre une synchronisation des mouvements articulaires en utilisant la méthode du temps minimal.

B) Espace opérationnel

Inspirez-vous de la fonction fournie *TrajectoireConsigne.m* afin de faire suivre à l'effecteur des trajectoires sous forme d'un cercle de rayon R = 2 et de centre C=(0, 7.5).

Partie II

COMMANDE DES ROBOTS ET EVITEMENT D'OBSTACLES

INTRODUCTION

Cette partie du TP introduit le modèle dynamique du robot SCARA 2ddl dans le but de traiter certaines problématiques liées à la commande des robots manipulateurs. Vous allez étudier plus particulièrement la commande du robot SCARA (RR) par PID ainsi que quelques notions de génération de trajectoires pour l'évitement d'obstacles.

IV RAPPEL ET PROGRAMMES FOURNIS

Le modèle dynamique d'un robot décrit un système de *n* équations différentielles du second ordre non linéaires et couplées (voir Annexe 1), *n* étant le nombre d'articulations. Pourtant, dans une commande classique, qui est celle de la plupart des robots industriels, le système robotique est considéré comme un système linéaire et chacune de ses articulations est asservie par une commande décentralisée de type PID à gains constants. Ses avantages sont la facilité d'implantation et le faible coût de calcul. En contrepartie, la réponse temporelle du robot variant selon sa configuration, on constate des dépassements de consigne et une mauvaise précision de suivi dans les mouvements rapides. Dans beaucoup d'applications, ces inconvénients ne représentent pas un gros handicap.

Dans ce type de commande les actions du PID sont les suivantes :

$$\begin{cases} K_{pj} = 3a_j\omega_j^2 \\ K_{dj} = 3a_j\omega_j \\ K_{Ij} = a_j\omega_j^3 \end{cases}$$

Avec :

- $a_j = A_{jj\,max}$ désigne la valeur maximale de l'élément A_{jj} de la matrice d'inertie du robot (cf. Annexe1),
- et $\omega_j > 0$ est choisi la plus grande possible, toutefois, cette pulsation ne devra pas être supérieure à la pulsation de résonance ω_{rj} du système mécanique afin de ne pas le déstabiliser (prendre $\omega_{rj} = 2.5$ rad/s).

Pour plus de détails sur la méthode d'obtention des actions du PID pour un robot manipulateur reportez-vous à l'Annexe 2.

Pour traiter cette partie du TP, un ensemble de programmes sont adjoints à ceux qui ont été précédemment détaillés dans la parie I. Les nouveaux programmes et leur contenu sont les suivants :

ModeleDynamiqueDirect.m	Définit l'accélération des articulations du robot SCARA en fonction des couples appliqués par ses actionneurs
SimulinkBibliothequeTP.mdl	Fichier Simulink contenant des blocs Simulink que vous pouvez directement utiliser dans votre TP
SimulinkControleRobotSCARA_AvecMD.mdl	Modèle Simulink (cf. Figure. 5) qui permet d'utiliser un régulateur de type PID pour commander le robot SCARA dans le domaine opérationnel

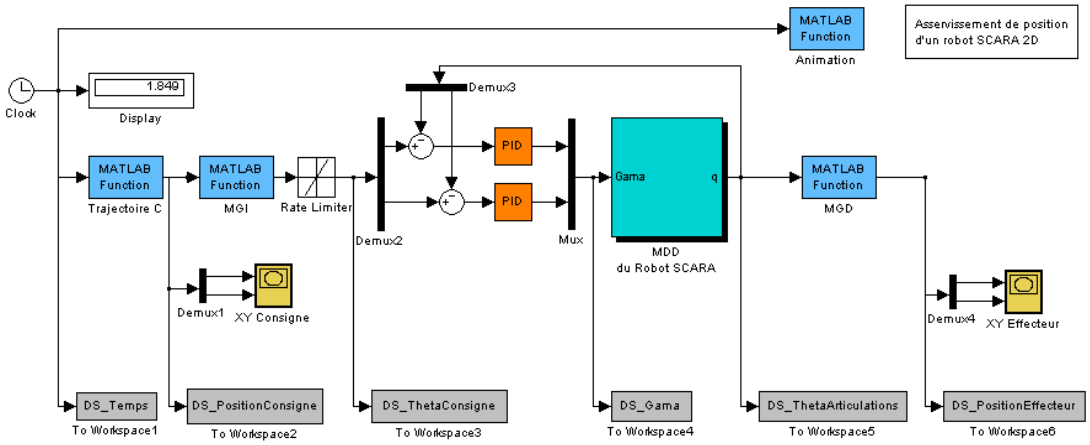


Figure. 5

V LE TRAVAIL A FAIRE

V.1 Commande du robot dans l'espace articulaire et opérationnel

1. Les paramètres actuels des PID (figure.5) ont été fixés empiriquement. Synthétisez les contrôleurs PID adaptés au robot SCARA en vous inspirant de l'Annexe 2. Une fois les paramètres des PID obtenus par programmation, vous devez automatiquement les affecter aux blocs correspondants.
   Indications :
   - cette partie de code doit apparaître dans le programme principal *SimulationRobotSCARA.m* et plus précisément entre « %% Début code PID %% » et « %% Fin code PID %% »,
   - inspirez-vous de la définition et de l'utilisation de la variable globale « T »,
   - utilisez la fonction MATLAB *fminbnd*.

2. Affichez dans une même figure les caractéristiques liées à la commande du robot SCARA pour suivre la trajectoire référence. Les éléments qui doivent être visibles sur la figure sont :
   - les coordonnées articulaires consignes et effectives des articulations,
   - les vitesses articulaires consignes et effectives des articulations,
   - les positions (x, y) consignes et effectives de l'organe terminal,
   - l'erreur de suivi de trajectoire,
   - les couples articulaires.

   Indication : cette partie de code est déjà en partie fournie, vous la trouverez dans le programme *GestionGUI.m* et plus précisément entre « %% Début code affichage figure %% » et « %% Fin code affichage figure %% ». Enlevez les commentaires et ajoutez la partie de code vous permettant de représenter l'évolution des vitesses consignes et effectives du robot.

3. Modifiez les fichiers *SimulinkControleRobotSCARA_AvecMD.mdl* et *GestionGUI.m* fournis afin de permettre de commander le robot dans le domaine articulaire via des interpolations polynomiales d'ordre 5. Nous souhaitons par ailleurs qu'aucune des articulations du robot ne puisse sortir en dehors de l'espace de travail du robot (cf. figure. 2) lors de l'exécution des mouvements. Réalisez le programme qui permet de mettre en place votre stratégie de contrôle.
   Indications :
   - modifiez le générateur de mouvement d'ordre 5 que vous avez réalisé dans la partie I afin de permettre d'affecter automatiquement les vecteurs q^i, q^f. q^i correspondra à la configuration courante du robot et q^f sera donnée directement à l'aide de la souris

- pour qu'aucune articulation ne sorte de l'espace de travail du robot, utilisez le programme *ModeleGeometriqueInverse_Bis.m* que vous avez développé précédemment. En l'occurrence exploitez la possibilité de pouvoir choisir quelle solution (Q1 (coude bas) ou Q2 (coude haut)) affecter au robot.

4. Est-ce que l'effecteur du robot atteint la configuration finale q^f exactement après le temps $t_f = \text{Max}\{t_{fj}\}$ fixé par le générateur de mouvement utilisé ? En modifiant l'influence des actions Proportionnelle, Dérivée et Intégrale de votre régulateur PID, essayez de réduire le temps de réponse du système robotique. Que concluez-vous ?

5. En utilisant les blocs Simulink fournis dans *SimulinkBibliothequeTP.mdl*, réalisez le schéma de contrôle donné en annexe (cf. figure. 7). Que constatez-vous à présent par rapport au temps de réponse du système ? Expliquez.

6. En gardant le régulateur utilisé en question 5 et en vous inspirant du schéma Simulink donné en Figure 3, réalisez la commande du robot SCARA dans son domaine opérationnel.
   Indication :
   - utilisez directement les consignes données par le programme *TrajectoireConsigne.m* fourni,
   - à défaut d'utiliser le MGI dans votre structure de contrôle utilisez le modèle cinématique inverse précédemment obtenu.

VI **EVITEMENT D'OBSTACLES** (GENERATION DE MOUVEMENT AVEC PASSAGE PAR DES POINTS INTERMEDIAIRES)

Dans certaines applications, les robots manipulateurs sont confrontés à la présence d'obstacles dans l'environnement. Ainsi, pour permettre à ces robots des mouvements sans collision avec les obstacles, il est impératif de les disposer d'une stratégie d'évitement d'obstacles appropriée.

On se propose dans ce qui suit d'étudier un cas simple où l'environnement est encombré avec deux obstacles circulaires (cf. Figure. 6) et qu'on veuille déplacer le robot de sa position initiale $P_i=[10,7\ 9,06]^T$ vers une position finale $P_f=[-1,8\ \ 12,9\ ]^T$.
Indication : Afin de permettre de visualiser les obstacles sur l'interface MATLAB, enlevez les commentaires entre « %% Début définition obstacles %% » et « %% Fin définition obstacles %% » existants dans le programme *SimulationRobotSCARA.m*.

1. Dans quel espace serait-il préférable de commander le robot pour éviter les obstacles présents dans l'environnement ? Commentez.

2. Proposez une méthode pour rallier la position P_f à partir de la position P_i tout en évitant les deux obstacles ? Implémentez et testez votre méthode. Commentez.
   Indication : Une fois initialisée votre méthode d'évitement d'obstacles, le robot doit réaliser sa trajectoire (de la position initiale à la position finale) sans s'arrêter.

Remarques concernant le compte-rendu et les programmes à fournir :

Votre compte-rendu devra faire référence aux différents outils de modélisation et de commande des robots manipulateurs que vous avez construits. Vous devez pour cela convenablement organiser l'ensemble de vos programmes afin qu'ils soient directement accessibles à la simulation. Par ailleurs, il est conseillé de rédiger l'aide correspondant à l'ensemble des fonctionnalités que vous avez programmées, cette Aide doit être affichée lors de l'appui sur le bouton poussoir «Aide» (Figure. 2).

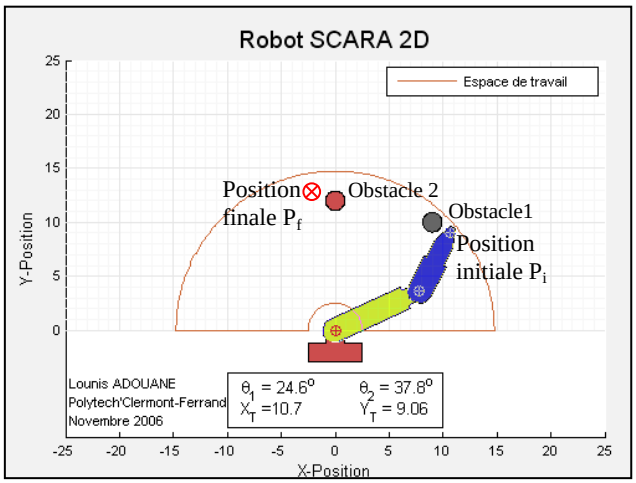


Figure. 6

ANNEXES

Annexe 1

Le formalisme de Lagrange est l'un des plus utilisés pour le calcul du modèle dynamique des robots. Ce formalisme décrit les équations du mouvement en terme de travail et d'énergie du système, ce qui se traduit, lorsque l'effort extérieur sur l'organe terminal est supposé nul, par l'équation (cas d'un robot à « *n* » ddl) :

$$\Gamma_i = \frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} \quad i = 1,..n$$

Avec :

- Γ_i : couple/force articulaire engendré par l'actionneur (articulation) *i*,
- *L* : Lagrangien du système égal à E – U,
- E : énergie cinétique totale du système,
- U : énergie potentielle totale du système.

Après simplification :

$$\Gamma = A(q).\ddot{q} + H(q,\dot{q})$$

Avec :

- *A* est la matrice (*n*×*n*) de l'énergie cinétique, elle est appelée aussi matrice d'inertie du robot, elle est symétrique et définie positive et ses éléments sont fonction des variables articulaires *q*.
- *H* est un vecteur caractéristique dépendant à la fois de *q* et de $\dot{q}$ (Khalil 99).

Les matrices *A* et le vecteur *H* sont fonction des paramètres géométriques et inertiels du robot. Les équations dynamiques d'un système mécanique articulé forment donc un système de *n* équations différentielles du second ordre, couplées et non linéaires.

Annexe 2

Le schéma classique d'une commande par PID d'un robot manipulateur est représenté sur la figure ci-dessous.

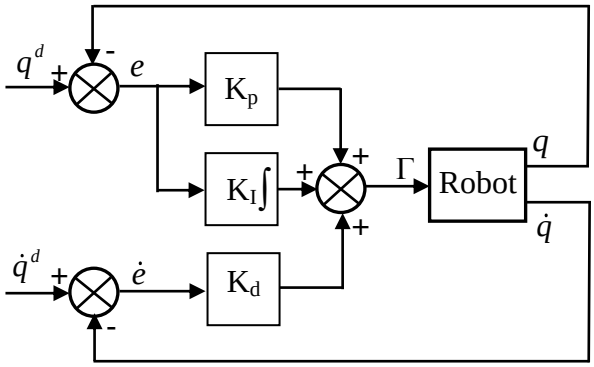


Figure. 7

La loi de commande du PID est donnée par :

$$\Gamma = K_p(q_d - q) + K_d(\dot{q}_d - \dot{q}) + K_I \int_{t_0}^{t} (q_d - q)dt$$

où $\dot{q}_d(t)$ et $q_d(t)$ désignent les vitesses et positions désirées dans l'espace articulaire et K_p, K_d *et* K_I sont des matrices diagonales définies positives, de dimension ($n \times n$), d'éléments génériques respectivement les gains proportionnels K_{pj}, dérivés K_{dj} et intégraux K_{Ij}.

Le calcul des gains K_{pj}, K_{dj} et K_{Ij} est effectué en considérant le modèle de l'articulation j représenté par le système linéaire du deuxième ordre à coefficients constants suivant :

$$\Gamma_j = a_j\ddot{q}_j + F_{vj}\dot{q}_j + \gamma_j$$

Équation dans laquelle $a_j = A_{jj\ max}$ désigne la valeur maximale de l'élément A_{jj} de la matrice d'inertie du robot, F_{vj} les frottements visqueux et γ_j représente un couple perturbateur.

En négligeant F_{vj} et γ_j la fonction de transfert du système en boucle fermée est donnée par :

$$\frac{q_j(s)}{q_{jd}(s)} = \frac{K_{dj}s^2 + K_{pj}s + K_{Ij}}{a_j s^3 + K_{dj}s^2 + K_{pj}s + K_{Ij}}$$

et l'équation caractéristique s'écrit donc :

$$\Delta(s) = a_j s^3 + K_{dj}s^2 + K_{pj}s + K_{Ij}$$

La solution la plus courante en robotique consiste à choisir les gains de manière à obtenir un pôle triple réel et négatif, ce qui donne la réponse la plus rapide possible sans oscillation. Par conséquent, l'équation caractéristique se factorise de la façon suivante :

$$\Delta(s) = a_j(s + \omega_j)^3$$

avec $\omega_j > 0$, et est choisi la plus grande possible, toutefois, cette pulsation ne devra pas être supérieure à la pulsation de résonance ω_{rj} du système mécanique pour ne pas le déstabiliser.

On en déduit finalement les gains : $\begin{cases} K_{pj} = 3a_j\omega_j^2 \\ K_{dj} = 3a_j\omega_j \\ K_{Ij} = a_j\omega_j^3 \end{cases}$.

COMMANDES USUELLES DE MATLAB

Tableau 1 : les commandes indispensables

help	donne toute la liste des aides
help *sujet*	affiche l'aide sur le *sujet*
commande ;	le ; permet d'interdire l'affichage du résultat de la commande à l'écran
quit, exit	Extinction de MATLAB

Tableau 2 : commandes sur fichiers et répertoires

cd *nomrépertoire*	change de répertoire de travail (cd .. : remonter d'un répertoire)
pwd	affiche le nom du répertoire courant
dir, ls	affiche le contenu du répertoire courant
delete *nomfichier*	supprime le fichier *nomfichier*
load *nomfichier*	import les variables du fichier *nomfichier.mat (format matlab)*
load *nomfichier.dat*	importe la matrice *nomfichier* du fichier *nomfichier.dat* (format texte)
load *nomfichier X Y*	importe uniquement les variables *X* et *Y* du fichier *nomfichier.mat*
nomfichier	exécute le script contenu dans le fichier *nomfichier.m* (commandes, définitions, variables)
save *nomfichier*	sauvegarde toutes les variables dans le fichier *nomfichier.mat*
save *nomfichier X Y*	sauvegarde uniquement les variables *X* et *Y* dans le fichier *nomfichier.mat*

Tableau 3 : gérer les variables

length(*A*)	retourne le nombre de composants du vecteur *A*
size(*A*)	retourne le nombre de lignes et de colonnes de la variable *A*
who	liste les variables utilisées
whos	donne la liste des variables ainsi que leur dimension
clear	supprime toutes les variables de l'espace de travail
clear *X Y*	supprime seulement les variables *X* et *Y*
= =, <, >, <=, >=	opérateurs de comparaison de variables (aide : *help eq, lt, gt, le, ge*)
x = y	affectation : x prend la valeur de y (aide : *help punct*)
a:b	intervalle [a, *a+1 … b*] (aide : *help colon*)
x=y(:,k)	le vecteur x est égal toute la $k^{ième}$ colonne de la matrice *y*
x=y(a:b,k)	le vecteur *x* est égal à la $k^{ième}$ colonne de la matrice y, mais uniquement du composant numéro *a* au composant numéro *b*
max(*X*)	donne le plus grand élément du vecteur *X*
max(*y(a :b,k)*)	donne le plus grand élément de la $k^{ième}$ colonne de la matrice y, mais uniquement de la ligne numéro *a* à la ligne numéro *b*
min(*X*)	si *X* est un vecteur, min(*X*) donne le plus petit élément de ce vecteur
min(*y(a :b,k)*)	donne le plus petit élément de la $k^{ième}$ colonne de la matrice y, mais uniquement de la ligne numéro *a* à la ligne numéro *b*

Tableau 4 : fonctions graphiques

plot(*X*,*Y*,'*cs*')	dessine le graphe de *Y* en fonction de *X* suivant la couleur *c* et en utilisant le symbole *s*. *c* peut prendre entre autres les valeurs : y (jaune), m (magenta), c (cyan), r (rouge), g (vert), b (bleu), w (blanc), b (noir) et i (invisible). *s* peut être entre autres : . (point), - (tiret), -. (tiret-point), + (plus), * (étoile), o (cercle), x (croix).
title('*texte*')	ajoute le *texte* comme titre de la figure
xlabel('*texte*')	ajoute le *texte* comme légende de l'axe des abscisses
ylabel('*texte*')	ajoute le *texte* comme légende de l'axe des ordonnées
gtext('*texte*')	placera le *texte* à partir du point sélectionné à la souris
legend('*texte1', 'texte2'...)*	place une légende sur la figure courante. texte1, texte2, etc. sont les légendes de différentes courbes.
axis([X_{min} X_{max} Y_{min} Y_{max}])	retrace la figure dans les limites imposées sur chaque axe
grid on *ou* grid off	ajoute *ou* enlève une grille sur la figure
figure	crée une nouvelle figure
figure(*num*)	la figure portant le numéro *num* devient la figure courante (celle ou seront dessinées les courbes). La figure est créée si elle n'existait pas.
hold on *ou* hold off	autorise *ou* interdit le dessin des prochaines fonctions sur la figure sans effacer les précédentes (hold off est le mode par défaut).
close	ferme la figure courante
close(*num*)	ferme la figure portant le numéro *num*
close all	ferme toutes les figures

Tableau 5 : programmation

x = input('text')	permet de saisir la valeur de *x* au clavier
disp*('xxx')*	affiche à l'écran la chaîne de caractères *xxx*
pause	attend un appui sur une touche quelconque du clavier avant d'exécuter l'instruction suivante.
for *i=min : max* … end	répétition en boucle. Pour *i = min* à *i = max* faire …
break	termine immédiatement le traitement d'une boucle for ou while
if *cond1* … elseif *cond2* … else … end	exécution conditionnelle. Si condition *cond1* faire … sinon si *cond2* faire …sinon faire…
C% *texte*	commentaires