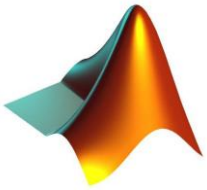

MEC6503
Utilisation de la fonction nR_MGD

La fonction MATLAB nommée **nR_MGB** permet de résoudre le **M**odèle **G**éométrique **D**irect d'un manipulateur sériel à **n** articulations **R**otoïdes.[1] Les exemples suivant montrent comment utiliser cette fonction.

Exemples d'utilisation

Aide de la fonction

On peut afficher l'aide de la fonction nR_MGD à l'aide de la commande :

```
>> help nR_MGD
```

On obtient alors l'aide suivante :

```
  Résolution du problème géométrique direct (MGD) pour un manipulateur
  sériel à n articulations rotoïdes (nR)

  Exemples d'appel de la commande :
    [T_EE,J] = nR_MGD([0, ... ,0])
    [T_EE,J] = nR_MGD()

  - Sans argument, la fonction retourne les coordonnées homogènes (T_EE) et
    la matrice Jacobienne (J) sous forme symbolique.
  - Avec [theta_1,...,theta_n] comme argument, la fonction retourne les
    coordonnées homogènes (T_EE) et la matrice Jacobienne (J) sous forme
    numérique

  !!! IMPORTANT : modifier les valeurs de 'theta', 'a', 'b', 'alpha' ci-bas
      pour réfléter les valeurs de votre manipulateur (lignes 33-36 et 50-52)

  Note : 'help nR_MGD' tapé à la ligne de commande affiche ces instructions
```

Modifications des paramètres du manipulateur dans le fichier nR_MGD.m

Pour pouvoir utiliser la fonction nR_MGD avec les paramètres propres à un manipulateur particulier, il est nécessaire de modifier certains paramètres dans le corps de la fonction :

`Theta` : Le vecteur $[\theta_1, \theta_2, \ldots, \theta_n]$ des angles θ_i de chacune des n articulations.

Les θ_i sont définis comme étant des variables symboliques à l'aide de la commande `sym('ti','real')`. Le vecteur `Theta` doit comporter autant de θ_i que le nombre d'articulations de votre manipulateur. Ce vecteur est utilisé pour la résolution symbolique du MGD lorsque la commande nR_MGD est appelée sans argument, c'est-à-dire lorsque la condition `(nargin == 0)` est respectée.

[1] Le code source de la fonction a été rédigé et commenté en se référant à :

Fundamentals of Robotic Mechanical Systems, Theory, Methods, and Algorithms; Series: Mechanical Engineering Series; Angeles, Jorge; 3rd ed., 2007, XXIV, 549 p., Hardcover; ISBN: 978-0-387-29412-4

`a` : Le vecteur $[a_1, a_2, \ldots, a_n]$ des paramètres a_i de Denavit-Hartenberg de chacune des n articulations

`b` : Le vecteur $[b_1, b_2, \ldots, b_n]$ des paramètres b_i de Denavit-Hartenberg de chacune des n articulations

`alpha` : Le vecteur $[\alpha_1, \alpha_2, \ldots, \alpha_n]$ des paramètres α de Denavit-Hartenberg de chaque articulation

Ces paramètres sont définis dans la section *Définition des constantes* du fichier nR_MGD.m :

```
%----------------------------------------
% Définitions des constantes (! À MODIFIER)
%----------------------------------------

    % Précision arithmétique désirée pour Maple (Affecte la commande 'vpa')
    digits(8);

    %------------------------
    % Si aucun argument (nargin == 0), retourne la jacobienne symbolique
    % où 't1' ... 'tn' représentent symboliquement les 'theta 1' à 'theta n'
    if (nargin == 0)
        theta = [sym('t1','real'),...  % ! AJOUTER OU SUPPRIMER CES LIGNES
                 sym('t2','real'),...  % AU BESOIN POUR QU'IL Y AIT AUTANT
                 sym('t3','real'),...  % DE LIGNES QUE D'ARTICULATIONS
                 sym('t4','real')];
    end

    %------------------------
    % Paramètres de Denavit-Hartenberg du robot (Voir Sec.:4.2)
    % ! À MODIFIER

    % Robot FANUC ARC MATE 120iL (joint 1 à 6 et outil)
%     a     = [  0.200,  0.800, -0.070,  0,      0,     0,      0         ];
%     b     = [  0.700,  0,      0,      0.810,  0,     0.100,  0.200     ];
%     alpha = [ -pi/2,   0,      pi/2,  -pi/2,   pi/2,  0,      60/180*pi ];

    % Robot 4R sphérique : Positio
    % (test : [T_EE,J] = nR_MGD([3*pi/4, pi/2, 3*pi/4, 3*pi/4] )
    a     = [      0;      0;  1; 2^0.5/2 ];
    b     = [      1;      0;  0;       0 ];
    alpha = [ (pi/2); (pi/2);  0;       0 ];
```

Résolution numérique

Une fois les modifications des paramètres apportées au fichier nR_MGD, il suffit d'invoquer la commande de la manière suivante afin de résoudre numériquement le MGD.

```
>> [T_EE,J] = nR_MGD([0,0,0,0])
```

Où : `[0,0,0,0]` : Est le vecteur $[\theta_1, \theta_2, \ldots, \theta_n]$ des angles θ_i de chacune des n articulations sous forme numérique

`T_EE` : Est la matrice de coordonnées homogènes numérique du bout du manipulateur

`J` : Est la matrice Jacobienne numérique

On obtient alors le résultat suivant pour le robot 4R sphérique Positio :

```
T_EE =

    1.0000         0         0    1.7071
         0   -1.0000   -0.0000         0
         0    0.0000   -1.0000    1.0000
         0         0         0    1.0000


J =

         0         0         0         0
         0   -1.0000   -0.0000   -0.0000
    1.0000    0.0000   -1.0000   -1.0000
         0         0         0         0
    1.7071    0.0000   -1.7071   -0.7071
         0    1.7071    0.0000    0.0000
```

Résolution symbolique

La résolution symbolique du MGD se fait de la même manière que pour la résolution numérique sauf que l'on omet l'argument `[0,0,0,0]` dans l'appel de la fonction :

```
>> [T_EE,J] = nR_MGD()
```

On obtient alors sous forme symbolique la matrice de coordonnées homogènes du bout du manipulateur ainsi que la matrice Jacobienne :

```
T_EE =

[...]


J =

[...]
```

Les solutions matricielles symboliques étant trop longues pour ce manipulateur, elles sont ici abrégées par `[...]`.

```
function [T_EE,J]=nR_MGD(theta)
%
% Résolution du problème géométrique direct (MGD) pour un manipulateur
% sériel à n articulations rotoïdes (nR)
%
% Exemples d'appel de la commande :
%   [T_EE,J] = nR_MGD([0, ... ,0])
%   [T_EE,J] = nR_MGD()
%
% - Sans argument, la fonction retourne les coordonnées homogènes (T_EE) et
%   la matrice Jacobienne (J) sous forme symbolique.
% - Avec [theta_1,...,theta_n] comme argument, la fonction retourne les
%   coordonnées homogènes (T_EE) et la matrice Jacobienne (J) sous forme
%   numérique
%
% !!! IMPORTANT : modifier les valeurs de 'theta', 'a', 'b', 'alpha' ci-bas
%     pour réfléter les valeurs de votre manipulateur (lignes 33-36 et 50-52)
%
% Note : 'help nR_MGD' tapé à la ligne de commande affiche ces instructions


%----------------------------------------
% Définitions des constantes (! À MODIFIER)
%----------------------------------------

    % Précision arithmétique désirée pour Maple (Affecte la commande 'vpa')
    digits(8);

    %------------------------
    % Si aucun argument (nargin == 0), retourne la jacobienne symbolique
    % où 't1' ... 'tn' représentent symboliquement les 'theta 1' à 'theta n'
    if (nargin == 0)
        theta = [sym('t1','real'),...  % ! AJOUTER OU SUPPRIMER CES LIGNES
                 sym('t2','real'),...  % AU BESOIN POUR QU'IL Y AIT AUTANT
                 sym('t3','real'),...  % DE LIGNES QUE D'ARTICULATIONS
                 sym('t4','real')];
    end

    %------------------------
    % Paramètres de Denavit-Hartenberg du robot (Voir Sec.:4.2)
    % ! À MODIFIER

    % Robot FANUC ARC MATE 120iL (joint 1 à 6 et outil)
%     a     = [  0.200,  0.800, -0.070,  0,      0,     0,      0         ];
%     b     = [  0.700,  0,      0,      0.810,  0,     0.100,  0.200     ];
%     alpha = [ -pi/2,   0,      pi/2,  -pi/2,   pi/2,  0,      60/180*pi ];

    % Robot 4R sphérique : Positio
    % (test : [T_EE,J] = nR_MGD([3*pi/4, pi/2, 3*pi/4, 3*pi/4] )
    a     = [      0;      0;  1; 2^0.5/2 ];
    b     = [      1;      0;  0;       0 ];
    alpha = [ (pi/2); (pi/2);  0;       0 ];



%----------------------------------------
% Corps de la fonction 'nR_MGD'
%----------------------------------------

    %------------------------
    % Calcul de A = [e_1 ... e_n] (Voir Eq.:5.7a p.170)
    A = ei(1);
    if length(theta)>1
        for u = 2:length(theta)
            A = [A,ei(u)];
        end
    end

    %------------------------
    % Calcul de B = [e_1 x r_1 ... e_n x r_n] (Voir Eq.:5.7b p.170)
    B = ei_ri(1);
    if length(theta)>1
        for u = 2:length(theta)
            B = [B,ei_ri(u)];
        end
    end
```

Suite :

```
    %------------------------
    % Calcul de la matrice Jacobienne J = [A;B] (Voir Eq.:5.10a et b)
    J = [[A];[B]];
    J = simplify(vpa(J));

    if not (nargin == 0)
        J = eval(J);
    end

    %------------------------
    % Calcul de la position et de l'orientation (coordonnées homogènes) du
    % dernier joint, soit celui du bout de l'outil (End-Effector) si n+1 joints
    T_EE = T(length(theta));
    T_EE = simplify(vpa(T_EE));

    if not (nargin == 0)
        T_EE = eval(T_EE);
    end




%----------------------------------------
% Définitions des fonctions
%----------------------------------------

    %------------------------
    % Fonction mui(i) %Voir Eq.:4.1a
    function val = mui(i)
        val = sin(alpha(i));
    end

    %------------------------
    % Fonction lambdai(i) %Voir Eq.:4.1a
    function val = lambdai(i)
        val = cos(alpha(i));
    end

    %------------------------
    % Fonction Q(i) : Permet de construire la matrice de rotation Qi
    % Voir Eq.:4.1e et Eq.:4.7
    function val = Qi(i)
        val = [[ cos(theta(i)),-lambdai(i)*sin(theta(i)), mui(i)*sin(theta(i))];...
               [ sin(theta(i)), lambdai(i)*cos(theta(i)),-mui(i)*cos(theta(i))];...
               [             0,                   mui(i),           lambdai(i)]];
    end

    %------------------------
    % Fonction P(i) : Permet de construire le vecteur position ai
    % (Voir Eq.:4.3b et Eq.:4.7 p.139)
    function val = ai(i)
        val = [[ a(i)*cos(theta(i))];...
               [ a(i)*sin(theta(i))];...
               [ b(i)]];
    end

    %------------------------
    % Fonction PP(i) : Permet de construire la matrice Pi(i) pour le calcul
    % de la matrice Jacobienne. (Voir p.176)
    function val = Pi(i)
       if     i == 1
           val = Qi(1);
       elseif i >  1
           val = Pi(i-1)*Qi(i); % Récursif
       end
    end
```

Suite :

```
    %------------------------
    % Fonction r(i) : Permet de construire le vecteur ri pour le calcul
    % de la matrice Jacobienne (Voir p.178)
    function val = ri(i)
       if     i == length(theta)
           val = ai(i);
       elseif i <  length(theta)
           val = ai(i)+Qi(i)*ri(i+1);  % Récursif
       end
    end

    %------------------------
    % Fonction e(i) : Permet de construire le vecteur ei pour le calcul
    % de la matrice Jacobienne (Voir p.175)
    function val = ei(i)
       if i == 1
           val = [0,0,1]'; % par définition : [e_1]_1 = [0,0,1]'
       else
           val = Pi(i-1)*[0,0,1]'; % [e_i]_1 est la dernière colonne de P_{i-1}
       end
    end

    %------------------------
    % Fonction e_r(i) : Permet de construire le produit vectoriel de ei(i)
    % et ri(i) rammené dans le repère 1 (Voir Eq.:5.34 p.179)
    % sachant que [e_i]_i = [0,0,1]'
    function val = ei_ri(i)
       if i == 1
           val = cross([0,0,1]',ri(i));
       else
           val = Pi(i-1)*(  cross([0,0,1]',ri(i))  );
       end
    end

    %------------------------
    % Fonction T(i) : Coordonnées du joint i dans le repère de la base
    % (Voir EQ.:4.10 p.140)
    function val = T(i)
       if     i == 1
           val = Ti(1);
       elseif i >  1
           val = T(i-1)*Ti(i); % Récursif
       end
    end

    %------------------------
    % Fonction Ti(i) : Permet de construire la transformée ti
    % (Voir p.140)
    function val = Ti(i)
        val = [[Qi(i),ai(i)];[0,0,0,1]];
    end




end

% Programme écrit par : Benoit Rousseau
%
% Référence:
%   Fundamentals of Robotic Mechanical Systems
%   Theory, Methods, and Algorithms;
%   Series: Mechanical Engineering Series;
%   Angeles, Jorge; 3rd ed., 2007, XXIV, 549 p., Hardcover;
%   ISBN: 978-0-387-29412-4
```