Sébastien Doyez
Lucien Reyser
MEA3
2022

Compte Rendu

Robotique de manipulation

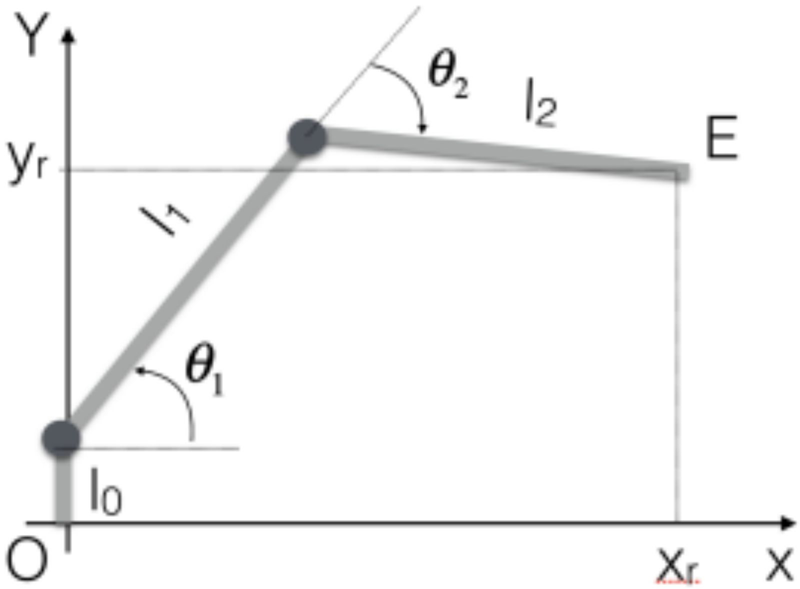
Sébastien Doyez
Lucien Reyser
MEA3
2022

Sommaire :

Sébastien Doyez
Lucien Reyser
MEA3
2022

Travail réalisé en binome par : Sébastien Doyez et Lucien Reyser

Déposé par :



Dans ce mini-projet, on considèrera la structure géométrique suivante :

• R1 = [l0 = 0.2m, l1 = 0.4m, l2 = 0.3m]

La première chose faite dans ce projet a été d' importer les librairies Plots et LinearAlgebra qui seront indispensables…

Puis dans un second temps il peut être intéressant de définir un certain nombre de variables redondantes en tant que variable globale :

```
# 1-Introduction :
global l0 = 0.2
global l1 = 0.4
global l2 = 0.3
global R=[l0 l1 l2];
global θ1=[0 pi];
global θ2=[-(pi/2) pi/2];
global VitesseθMax=[5 5];
global Vitesse=[1 0]
```

1-Modélisation Géométrique

Sébastien Doyez
Lucien Reyser
MEA3
2022

1.1 Modèle Géométrique Direct (MGD)

Ecrire le modèle géométrique direct du bras manipulateur figure (1) en exprimant :

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x_r} \\ \boldsymbol{y_r} \end{pmatrix} = \boldsymbol{f(\theta)}$$

Déterminer la région accessible par l'effecteur du robot manipulateur (xr, yr) en considérant

ϑ1 = [0, π], ϑ2 = [− π 2 , π 2]. Tracer la surface de travail accessible.

Modèle Géométrique Direct (MGD)

- On peut exprimer $X = \begin{pmatrix} x_r \\ y_r \end{pmatrix} = \begin{pmatrix} l_1\cos(\theta_1) + l_2\cos(\theta_1+\theta_2) \\ l_1\sin(\theta_1) + l_2\sin(\theta_1+\theta_2) \end{pmatrix}$.

En effet, la MGD est traduite par l'équation ci-dessus. De ce fait, on déclare une fonction « MGD » qui prend en argument, deux angles qui correspondent aux angles des bras du robot. Cette fonction aura pour but de renvoyer la position du bras avec des coordonnées cartésiennes.

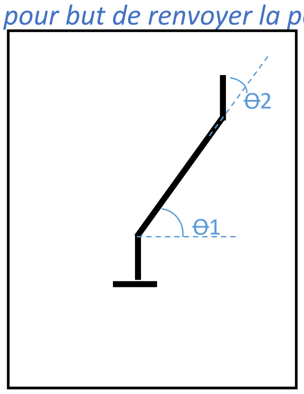


Schéma des angles du robot

Cependant, dans le cas de notre robot, on doit prendre en compte le segment l0 . Ainsi on a :

$$\begin{pmatrix} x_r \\ y_r \end{pmatrix} = \begin{pmatrix} l1\cos(\theta1)+l2\cos(\theta1+\theta2) \\ l0+l1\sin(\theta1)+l2\sin(\theta1+\theta2) \end{pmatrix}$$

En effet, on doit rajouter à la coordonnée yr le segment fixe l0 qui n'aura d'impact que sur la coordonnée y

Ainsi on aura le code suivant :

```
#2-Modelisation Géométrique :
#2.1 Modèle Géométrique Direct (MGD)

function MGD(θ1,θ2)

    x=R[2]*cos.(θ1)+R[3]*cos.(θ1+θ2);
    y=R[1]+R[2]*sin.(θ1)+R[3]*sin.(θ1+θ2);
    z=[x,y];
    return z;
end
```

Maintenant qu'on a une fonction qui nous permet d'obtenir des coordonnées cartésiennes à partir des angles du bras, on peut tracer la surface de travail accessible… Pour se faire, on déclare une fonction tel que :

```
function acces_Rob()
    N=50;
    t1=θ1[1]:θ1[2]/N:θ1[2];
    t2=θ2[1]:(θ2[2]*2)/N:θ2[2];
    x=zeros(N,N);
    y=zeros(N,N);
    for i=1:N
        for j=1:N
            x[i,j],y[i,j]=MGD(t1[i],t2[j]);
        end
    end

    figure = scatter(x,y,legend=false);
    plot(figure)

end
```

Dans cette fonction on va prendre une variable N qui aura pour but de définir le pas. Puis on déclare 2 variables : t1 et t2, 2 tableaux qui vont prendre respectivement l'angle minimum que peut parcourir le segment l1 à son angle maximum avec un pas de :

angle maximum/N

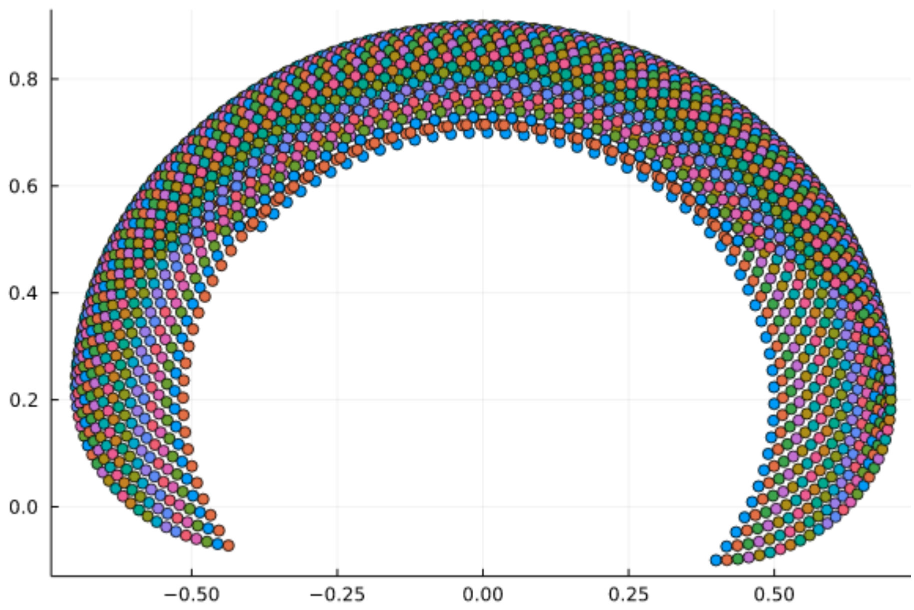
Pour t2, l'angle minimum que peut parcourir le segment l2 à son angle maximum avec un pas de :

2 angle maximum/N*

pour avoir un tableau de la même taille que t1.

Puis on va déclarer 2 matrices x et y qu'on va remplir de 0. Une fois ceci fait, on parcourt une double boucle qui aura pour but de remplir la matrice x et y des données équivalentes aux N angles que le bras peut prendre, rangées dans les tableaux t1 et t2 en coordonnées cartésiennes via l'appel à la fonction MGD déclaré précédemment. Et enfin, on trace ces points, ce qui nous donne la surface de travail accessible.

La surface de travail accessible obtenue est :



Graphe de la surface de travail accessible

Sébastien Doyez
Lucien Reyser
MEA3
2022

1.2 Modèle Géométrique Inverse (MGI)

Ecrire le modèle géométrique inverse du bras manipulateur figure (1) en exprimant :

$$\begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix} = g(X)$$

On considèrera les 2 solutions possible du modèle : coude haut et coude bas. Réaliser un programme utilisant le MGI pour vérifier la validité des résultats précédents (espace de travail). Proposer une méthode permettant de vérifier si la position (xr, yr) est accessible. Tracer la solution faisant apparaitre les positions atteignables coude haut, coude bas, les 2.

Concernant la MGI, on a deux fonctions : Une première consiste à donner un point en paramètre, celle-çi nous indique si le point est accessible ou non et si c'est le cas, elle nous donne une représentation du robot dans les configurations accessibles (cf figure 2.2.1, 2.2.2 et 2.2.3).

La seconde nous donne tous les points accessibles par le robot en coude haut (cf figure 2.2.4), en coude bas (cf figure 2.2.5) et dans les deux configurations. (cf figure 2.2.6)

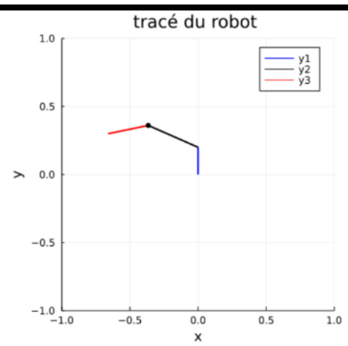


Figure 2.2.1 Coude bas

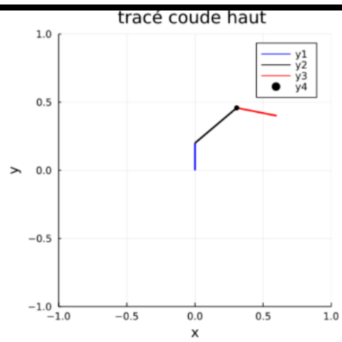


Figure 2.2.2 Coude haut

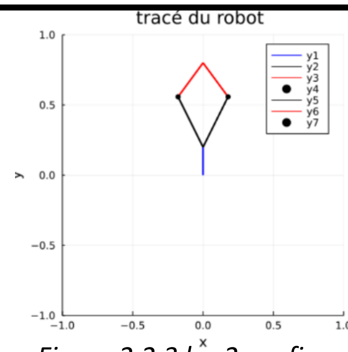


Figure 2.2.3 les 2 configurations

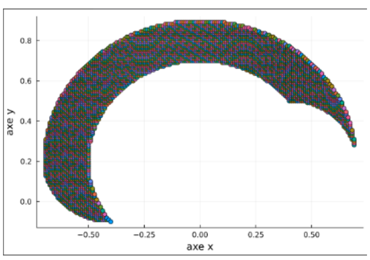


Figure 2.2.5 Coude bas

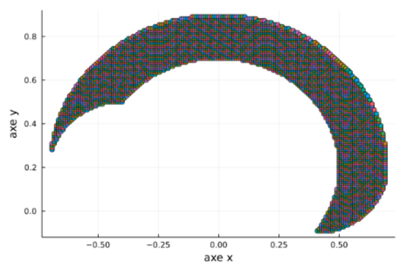


Figure 2.2.4 Coude haut

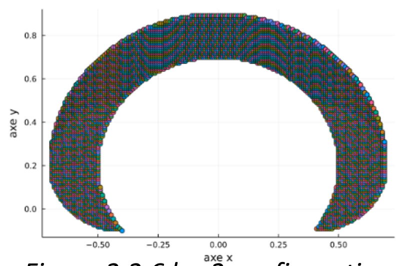


Figure 2.2.6 les 2 configurations

Sébastien Doyez
Lucien Reyser
MEA3
2022

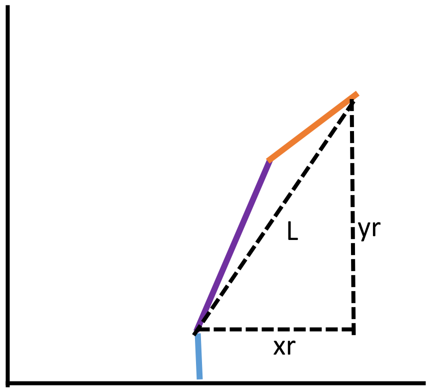
```
function MGI(x,y,Coude)
    y=y-R[1]
    i=(x^2+y^2-(R[2]^2+R[3]^2))/(2*R[2]*R[3])
    if(abs(i)<=1)
        if(acos(i)>=0)
            if(Coude=="Coude bas")
                θ2=acos(i)
            else
                θ2=-acos(i)
            end
        else
            if(Coude=="Coude bas")
                θ2=-acos(i)
            else
                θ2=acos(i)
            end
        end
        if(abs(cos(θ2))<=1)
            b=(x*(R[2]+R[3]*cos(θ2))+y*R[3]*sin(θ2))/(x^2+y^2)
            c=(y*(R[2]+R[3]*cos(θ2))-x*R[3]*sin(θ2))/(x^2+y^2)
            θ1=atan(c,b)
            if(verification(θ1)==1)
                if((θ1>=0 && θ1<=pi) && (θ2>=-pi/2 && θ2<=pi/2))
                z=[θ1,θ2]
                return z

            else
                    return "inaccessible"
                end
            else

                return "inaccessible"
            end
        else

            return "inaccessible"
        end
    else
        return "inaccessible"
    end
end
```

Dans un premier temps, on a la fonction MGI qui prend en paramètre une coordonnées x et y et la configuration dans laquelle on veut le robot (Coude haut ou Coude bas) . Celle-çi nous renverra la valeur que doivent prendre les angles theta1 et theta2 pour se retrouver à la coordonnée voulue …

Pour se faire, on calcule tout d'abord theta2 à l'aide de la fonction :

$$\theta_2 = \pm \arccos\left(\frac{x_r^2 + y_r^2 - (l_1^2 + l_2^2)}{2l_1 l_2}\right)$$

Cependant dans notre cas, il faut retrancher la hauteur du segment l0=0.2m qui est fixe et donc qui n'intervient pas dans le calcul des angles theta1 et theta2. Il ne faut pas oublier de vérifier que la valeur absolue de cos(theta2) et bien supérieure ou égale à 1 car au-delà, l'arccosinus n'est pas défini. Ceci étant fait, si l'on souhaite que le robot aie une configuration coude bas, si le résultat est supérieur à 0 dans ce cas on définit :



$\theta_2 = \text{acos}\left(\frac{x^2+y^2-(l_1^2+l_2^2)}{2l_1l_2}\right)$ *. Sinon, si on veut que theta2 soit en coude bas,*

On doit prendre theta2 tel que : $\theta_2 = -\text{acos}\left(\frac{x^2+y^2-(l_1^2+l_2^2)}{2l_1l_2}\right)$ *. A contrario,*

si l'on veut que le robot soit en configuration coude haut on reprend

le même principe en inversant le signe de thêta 2.

Ensuite, on définit cosinus theta1 et sinus thêta 1 tel que :

$$sin\theta1 = \frac{y_r(l_1 + l_2\cos(\theta2)) - x_r l_2 \sin(\theta2)}{(x_r^2 + y_r^2)}$$

$$cos\theta1 = \frac{x_r(l_1 + l_2\cos(\theta2)) - y_r l_2 \sin(\theta2)}{(x_r^2 + y_r^2)}$$

Donc, nous avons theta1 qui est égale à arctangente de sin(theta1)/cos(theta1). Il ne reste plus qu'à vérifier d'une part que la position est atteignable via la fonction vérification qui a pour but de renvoyer 1 si la valeur absolue de cos(x) et sin(x) est inférieure ou égale à 1 et d'autre part, regarder que l'angle thêta 1 est bien entre 0 et pi et thêta 2 entre –pi/2 et pi/2. Après chaque vérification, si la position n'est pas accessible, la fonction retourne « inaccessible ».

Sébastien Doyez
Lucien Reyser
MEA3
2022

Comme nous l'avons vu précédemment pour tracer les points, on a 2 fonctions : Une première consiste à donner un point en paramètre ; celle-ci nous indique si le point est accessible ou non et si c'est le cas, il nous donne une représentation du robot dans les configurations accessibles :

```
function affichage_coude_bas(   x ,y)
    if (MGI(x,y,"Coude bas") != "inaccessible")
        # ici on travaille avec le coude bas:
        l0=R[1];
        l1=R[2];
        l2=R[3];
        mgi=MGI(x,y,"Coude bas");
        θ1_bas=mgi[1];
        θ2_bas=mgi[2];
        figure4=plot(title="tracé du robot ",xlabel="x", ylabel="y",label="",xlim=(-1,1),ylim=(-1,1), aspect_ratio=:e
        plot!(figure4,[0;0],[0;l0],color="blue",linewidth=2)

        ext1_bas=[l1*cos(θ1_bas);l0+l1*sin(θ1_bas)];
        x1_bas=[0;ext1_bas[1]];
        y1_bas=[l0;ext1_bas[2]];
        plot!(figure4,x1_bas,y1_bas,color="black",linewidth=2)

        ext2_bas=[ext1_bas[1]+l2*cos(θ1_bas+θ2_bas);ext1_bas[2]+l2*sin(θ1_bas+θ2_bas)];
        x2_bas=[ext1_bas[1];ext2_bas[1]];
        y2_bas=[ext1_bas[2];ext2_bas[2]];
        plot!(figure4,x2_bas,y2_bas,color="red",linewidth=2)

        plot!(figure4,[ext1_bas[1]],[ext1_bas[2]],seriestype=:scatter,color="black",label="")
        display(figure4)
    else
        print("cette position est inaccessible en Coude bas")
    end
```

La fonction affichage coude bas prends en paramètre une coordonnée x et y. On calcule le theta1 et theta2 de ce point via la fonction MGI définie au préalable qui elle, va prendre en paramètre x,y et le coude qui nous intéresse, donc, coude bas dans notre cas. Avec la fonction plot! qui est définit dans le module « Plots », on trace la droite d'origine x1=0 y1=0 et x2=0 y2=0.2

Puis via les theta1 et theta2 obtenues, on les retransforme en x et y via les formules :

$$x_{r1} = l_1 \cos(\theta 1)$$

$$y_{r1} = l_1 \sin(\theta 1) + l_0$$

Sans oublier d'ajouter l0 à yr .

Une fois ceci fait, on rajoute sur le plot la droite : x2=0 y2=0.2 à x3=xr1 et y3=yr1 qui n'est ni plus ni moins que la partie du bras l1. Et enfin on calcul les coordonnées x,y du derniers segment du robot via les formules :

$$x_{r2} = l_1 \cos(\theta 1) + l_2 \cos(\theta 1 + \theta 2)$$

$$y_{r2} = l_1 \sin(\theta 1) + l_2 \sin(\theta 1 + \theta 2) + l_0$$

Et donc on trace le dernier segment qui est représenté par la droite : x3=xr1 y3=yr1 à x4=xr2 y4=yr2.

Dans le cas ou la position voulue n'est pas accessible, on renvoie « cette position est inaccessible en coude bas ».

Sébastien Doyez
Lucien Reyser
MEA3
2022

Pour la fonction « affichage_coude_haut », on reprend le même principe que pour affichage coude bas hormis la fonction MGI qui, dans ce cas va prendre en paramètre le point x,y et « Coude haut » pour calculer la valeur (si cette position est accessible) de thêta 1 et thêta 2.

```
function affichage_coude_haut(   x, y)
    if (MGI(x,y,"Coude haut") != "inaccessible")
        #ici on travaille en coude haut :
        mgi = MGI(x,y,"Coude haut");
        l0=R[1];
        l1=R[2];
        l2=R[3];
        θ1_haut=mgi[1];
        θ2_haut=mgi[2];
        #initialisation de la figure:
        figure4=plot(title="tracé coude haut ",xlabel="x", ylabel="y",label="",xlim=(-1,1),ylim=(-1,1), aspect_ratio=
        plot!(figure4,[0;0],[0;l0],color="blue",linewidth=2)

        # définissons les extremitées:
        ext1_haut=[l1*cos(θ1_haut);l0+l1*sin(θ1_haut)];
        x1_haut=[0;ext1_haut[1]];
        y1_haut=[l0;ext1_haut[2]];
        plot!(figure4,x1_haut,y1_haut,color="black",linewidth=2)
        ext2_haut=[ext1_haut[1]+l2*cos(θ1_haut+θ2_haut);ext1_haut[2]+l2*sin(θ1_haut+θ2_haut)];
        x2_haut=[ext1_haut[1];ext2_haut[1]];
        y2_haut=[ext1_haut[2];ext2_haut[2]];
        plot!(figure4,x2_haut,y2_haut,color="red",linewidth=2)

        plot!(figure4,[ext1_haut[1]],[ext1_haut[2]],seriestype=:scatter,color="black")
        display(figure4)
    else
        print("Cette position est inaccessible en Coude haut")
    end
end
```

La fonction « affichage_duo(x , y) » quant à elle, associe les deux fonctions vues précédemment afin de tracer simultanément les deux configurations possibles (si la position est accessible) du robot pour un point.

```
function affichage_rob(R,x,y)
    if ((MGI(x,y,"Coude bas") != "inaccessible")&& (MGI(x,y,"Coude haut") != "inaccessible"))
        affichage_duo(R,x,y);
        print("la position est accessible en coude haut et bas")
    elseif ((MGI(x,y,"Coude bas") != "inaccessible")&& (MGI(x,y,"Coude haut") == "inaccessible"))
        affichage_coude_bas(R,x,y);
        print("la position n'est accessible qu'en coude bas");
    elseif ((MGI(x,y,"Coude bas") == "inaccessible")&& (MGI(x,y,"Coude haut") != "inaccessible"))
        affichage_coude_haut(R,x,y);
        print("la position n'est accessible qu'en coude haut");
    else
        print("la position est inaccessible....")
        return 0
    end
end
```

Et enfin la fonction affichage_rob(x , y) a pour but de simplifier l'utilisation des trois fonctions çi-dessus. En effet on rentre la coordonnée souhaitée et celle-çi trace toutes les configurations possibles

POLYTECH MONTPELLIER

Sébastien Doyez
Lucien Reyser
MEA3
2022

La seconde fonction quant à elle, trace tous les points accessibles en coude haut, en coude bas ou dans les deux configurations à travers un graphe de points (cf Figure 2.2.4, 2.2.5 et 2.2.6 ci-dessus)

En effet cette fonction prend en paramètre la configuration du bras dans laquelle on souhaite afficher les points. On définit deux tableaux, un coordonnées x et l'autre de coordonnées y qui vont de

-la taille max du bras à +la taille max du bras afin de traiter tout l'espace accessible par celui-çi sans avoir à le majorer de façon excessive. Puis en fonction de la configuration demandée, on va calculer successivement tous les point définis dans les deux tableaux de coordonnées, en faisant intervenir la MGI dans un premier temps pour avoir un thêta 1 et un thêta 2 si le point est accessible dans la configuration demandée, puis les repasser en coordonnée cartésienne via la fonction MGD définie çi-dessus (1.1 Modèle Géométrique Direct) afin de les tracer sur le figure

```
function AccesMGI(Coude)
    taille=R[1]+R[2]+R[3]
    N=30
    fig= plot(0,0,title="MGI",xlabel="axe x", ylabel="axe y")
    x=-taille:taille/N:taille
    y=-taille:taille/N:taille
    if (Coude=="Coude haut")
        for i=1:(2*N)
            for j=1:(2*N)
               if MGI(x[i],y[j],"Coude haut") != "inaccessible"
                    theta1,theta2=MGI(x[i],y[j],"Coude haut")
                    w,z=MGD(theta1,theta2)
                    fig=scatter!([w],[z],legend=false)
                end
            end
        end
    else
        if (Coude=="Coude bas")
            for i=1:(2*N)+1
                for j=1:(2*N)+1
                    if (MGI(x[i],y[j],"Coude bas")!="inaccessible")
                        theta1,theta2=MGI(x[i],y[j],"Coude bas")
                        w,z=MGD(theta1,theta2)
                        fig=scatter!([w],[z],legend=false)
                    end
                end
            end
        else
            for i=1:(2*N)+1
                for j=1:(2*N)+1
                    if (MGI(x[i],y[j],"Coude haut")!="inaccessible" && MGI(x[i],y[j],"Coude bas")!="inaccessible" )
                        theta1,theta2=MGI(x[i],y[j],"Coude haut")
                        w,z=MGD(theta1,theta2)
                        fig=scatter!([w],[z],legend=false)
                        theta1,theta2=MGI(x[i],y[j],"Coude bas")
                        w,z=MGD(theta1,theta2)
                        fig=scatter!([w],[z],legend=false)
                    else
                        if (MGI(x[i],y[j],"Coude bas")!="inaccessible")
                        theta1,theta2=MGI(x[i],y[j],"Coude bas")
                        w,z=MGD(theta1,theta2)
                        fig=scatter!([w],[z],legend=false)

                        else
                            if (MGI(x[i],y[j],"Coude haut")!="inaccessible")
                                theta1,theta2=MGI(x[i],y[j],"Coude haut")
                                w,z=MGD(theta1,theta2)
                                fig=scatter!([w],[z],legend=false)
                            end
                        end
                    end
                end
            end
        end
    end
    display(fig)
end
```

Sébastien Doyez
Lucien Reyser
MEA3
2022

2-Modélisation Cinématique

2.1 Modèle Cinématique Direct (MCD)

Dériver l'équation du MGD tel que :

$$\dot{\boldsymbol{X}} = \boldsymbol{J}.\dot{\boldsymbol{\theta}}$$

Donner l'expression de la matrice J. En considérant les vitesses maximales dans l'espace articulaire $|\dot{\theta}_{1max}|$ = 5rd/s et $|\dot{\theta}_{2max}|$ = 5rd/s, représenter le champ des vecteurs vitesses cartésiennes dans l'espace de travail.

Le modèle cinématique directe est la dérivée du modèle géométrique direct. Nous avions pour le modèle géométrique direct l'expression suivante :

$$\begin{pmatrix} x_r \\ y_r \end{pmatrix} = \begin{pmatrix} l1\, cos(\theta 1) + l2cos\, (\theta 1 + \theta 2) \\ l0 + l1\, sin(\theta 1) + l2sin\, (\theta 1 + \theta 2) \end{pmatrix}$$

Ainsi par opération de dérivée, nous obtenons l'expression suivante pour le modèle cinématique directe :

$$\begin{pmatrix} \dot{x}_r \\ \dot{y}_r \end{pmatrix} = \begin{pmatrix} -l1\, sin(\theta 1)\, \dot{\theta 1} - l2sin\, (\theta 1 + \theta 2)\dot{\theta 1} - l2sin\, (\theta 1 + \theta 2)\dot{\theta 2} \\ l1\, cos(\theta 1)\dot{\theta}_1 + l2cos(\theta 1 + \theta 2)\dot{\theta}_1 + l2cos(\theta 1 + \theta 2)\dot{\theta}_2 \end{pmatrix}$$

Nous avons donc par identification :

$$J = \begin{bmatrix} -l1\, sin(\theta 1) - l2sin\, (\theta 1 + \theta 2) & -l2sin\, (\theta 1 + \theta 2) \\ l1\, cos(\theta 1) + l2cos(\theta 1 + \theta 2) & l2cos(\theta 1 + \theta 2) \end{bmatrix}$$

Ainsi nous obtenons la fonction suivante qui prend en paramètre thêta 1 et thêta 2 et renvoie la matrice Jacobienne associée :

Sébastien Doyez
Lucien Reyser
MEA3
2022

```
function Jacob_MCD(θ1,θ2)
    t1_1 = -R[2]*sin(θ1)-R[3]*sin(θ1+θ2)
    t1_2 = -R[3]*sin(θ1+θ2)
    t2_1 = R[2]*cos(θ1)+R[3]*cos(θ1+θ2)
    t2_2 = R[3]*cos(θ1+θ2)
    jacobien=[ t1_1 t1_2 ; t2_1 t2_2 ]

    return jacobien;
end
```

En effet, on définit chacun des thermes de la matrice, puis on forme une matrice carrée 4x4 qu'on stocke dans la variable « jacobien »

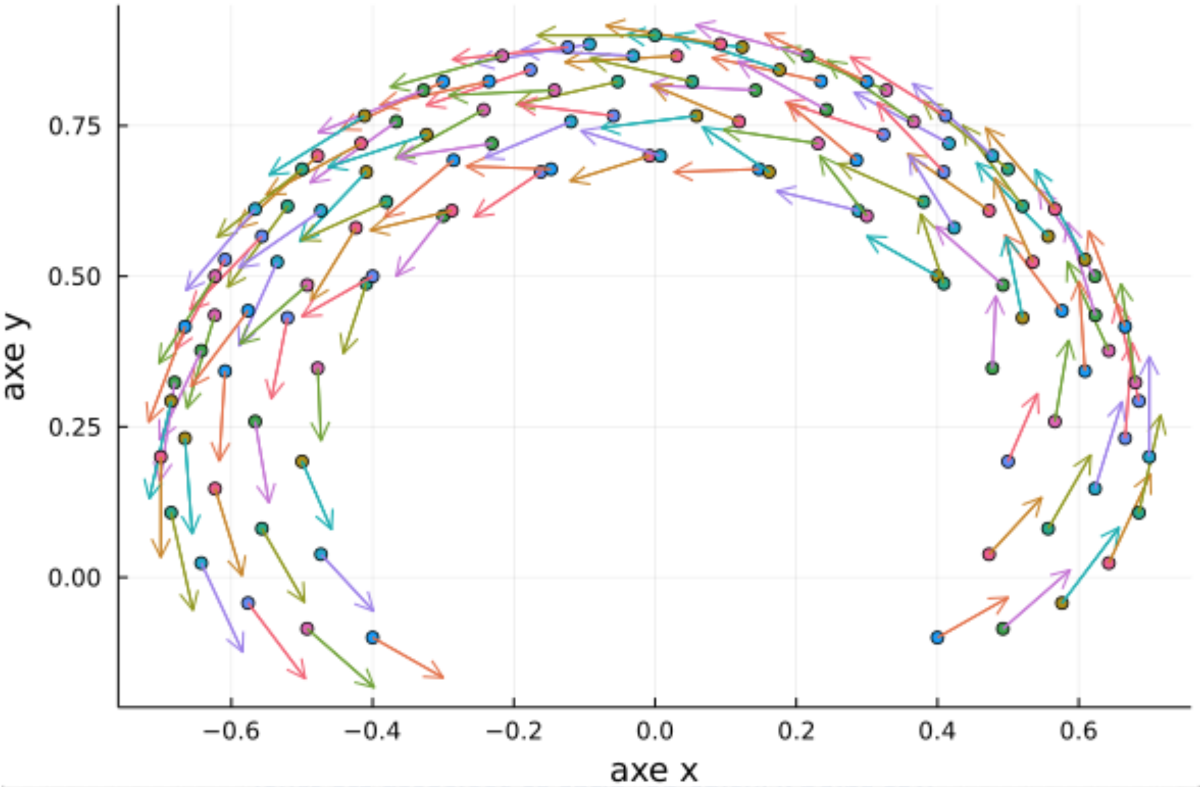
Pour représenter le champ des vecteurs vitesses cartésiennes dans l'espace de travail, on a défini une fonction « vectMCD() » :

Cette fonction définit deux tableaux t1 et t2. Le premier est un tableau qui prend les valeurs de thêta 1 minimum au maximum avec un pas qui est modulé par la variable N. Le deuxième va de thêta 2 minimum au maximum également par pas régulé par N.

```
function vectMCD()
    fig= plot(0,0,title="vecteur_MCD",xlabel="axe x", ylabel="axe y")
    N=10
    t1=θ1[1]:θ1[2]/N:θ1[2]
    t2=θ2[1]:θ2[2]/(N/2):θ2[2]
    for i=1:N+1
        for j=1:N+1
                    Point=MGD(t1[i],t2[j])
                    jacob=Jacob_MCD(t1[i],t2[j])
                    x_der = jacob[1,1]*5+jacob[1,2]*5
                    y_der = jacob[2,1]*5+jacob[2,2]*5
                    fig=scatter!([Point[1]],[Point[2]],legend=false)
                    fig=quiver!([Point[1]],[Point[2]],quiver=([x_der/30],[y_der/30]))
                end
            end
        end
    end
    display(fig)
end
```

Grace à deux boucles, on parcourt chacun des points des deux tableaux pour lesquelles on calcul leurs coordonnées cartésiennes grâce à la fonction MGD. On calcul la matrice jacobienne qui leurs est associées et enfin, on calcul x_point et y_point en multipliant la vitesse angulaire maximum par la matrice jacobienne. Puis nous traçons les points avec leurs vecteurs associés via les fonctions scatter et quiver du module « Plots ». Cependant, on divise les vecteurs obtenus par 30 afin de ne pas avoir une représentation déformée, due à des vecteurs relativement grands en comparaison avec l'espace de travail du bras robotisé. On obtient donc la figure ci-dessous :

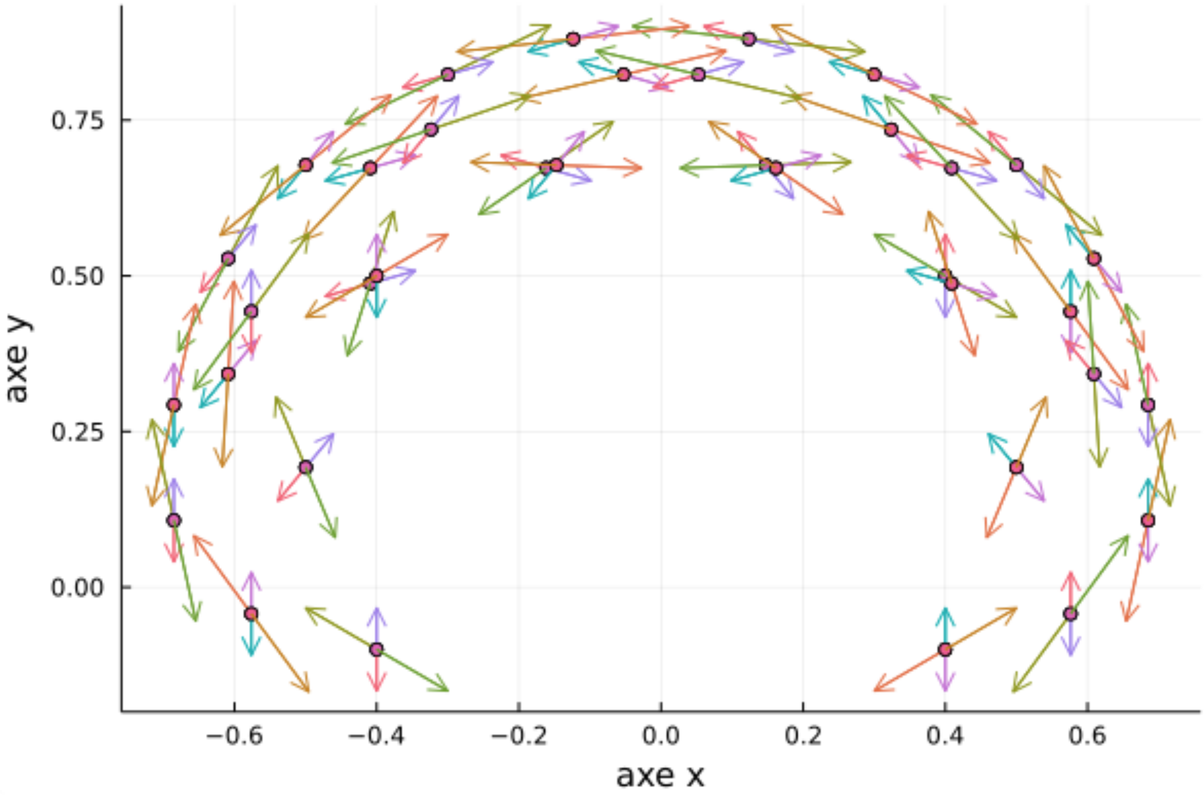
Sébastien Doyez
Lucien Reyser
MEA3
2022



Graphe de la représentation du champ des vecteurs vitesses cartésiennes dans l'espace de travail

Pour aller plus loin, nous avons un peu modifié la fonction, afin de tracer les vecteurs vitesses pour les vitesses angulaires : $\dot{\theta}_1 = \begin{cases} -5rad/s \\ 5rad/s \end{cases}$ *et* $\dot{\theta}_2 = \begin{cases} -5rad/s \\ 5rad/s \end{cases}$ *ce qui nous donne donc 4 vecteurs par point. On obtient alors la figure ci-dessous :*

Sébastien Doyez
Lucien Reyser
MEA3
2022



Graphe de la représentation du champ des vecteurs vitesses cartésiennes dans l'espace de travail

Avec vitesses angulaires différentes

2.2 Modèle Cinématique Indirect (MCI)

En utilisant l'équation précédente, exprimer le vecteur vitesse articulaire tel que :

$$\dot{\theta} = J^{-1}\dot{X}$$

Exprimer les conditions de validité de cette équation. Choisir un point de départ et un point d'arrivée dans l'espace opérationnel. En considérant une vitesse cartésienne $\dot{x}$ =1m.s^{-1},

$\dot{y}$ = 0m.s^{-1} tracer les positions et vitesses articulaires lors du déplacement du point de départ au point d'arrivée.

Respecte-t-on les contraintes articulaires du robot ?

Sébastien Doyez
Lucien Reyser
MEA3
2022

Sur la trajectoire que vous avez choisie, trouvez la vitesse linéaire maximum permettant de respecter les contraintes du robot.

L'une des conditions primordiales pour que l'équation soit valide est l'inversibilité de la matrice jacobienne. Il est donc indispensable de vérifier avant toute chose, que le déterminant de la matrice est bien différent de 0.

Ainsi, on déclare une fonction MCI qui prend en paramètre d'entrée theta1 et theta2. On calcul la matrice jacobienne qui leurs est attribué grâce à la fonction « Jacob_MCD » et on vérifie si elle est inversible. Si ce n'est pas le cas, la fonction renvoie « matrice non inversible ». Dans le cas contraire, on inverse la matrice jacobienne et on la multiplie par les vitesses cartésiennes voulues (qui dans notre cas, sont déclarées en tant que variables globales) ce qui nous donne donc les vitesses angulaires qu'on retourne à travers la variable « z ».

```
function MCI(θ1,θ2)
    Jacob=Jacob_MCD(θ1,θ2)
    if(abs(det(Jacob))>0.0000001)
Pour eviter les 0E-16 ect...
        J_inv=inv(Jacob)
        z= J_inv * Vitesse';
        return z
    else
        return "matrice non inversible"
    end
end
```

Le but ici est de réaliser deux courbes, pour nous permettre de mieux visualiser la position et la vitesse angulaire :

Pour se faire, nous avons tout d'abord initialiser 4 matrices qui stockera la position et la vitesse angulaire de nos ϑ1 et ϑ2… puis on a réalisé une boucle for afin de faire aller le bras de sa position initial à la position final choisi. Il a fallu faire attention au fait qu'à tout moment une position pouvait ne pas être accessible, et cela rendrait notre trajectoire impossible. C'est pour cela que nous avons utilisé la fonction « possible » qui va nous indiquer si la position (xdep +dx , y) est atteignable (celle-ci passera à 0 si c'est impossible).

```
function possible(R,x,y)
    ## 0 impossible
    ## 1 => CH + CH
    ##2 CB
    ## 3 CH
    if ((MGI(x,y,"Coude bas") != "inaccessible")&& (MGI(x,y,"Coude haut") != "inaccessible"))
        return 1;
    elseif ((MGI(x,y,"Coude bas") != "inaccessible")&& (MGI(x,y,"Coude haut") == "inaccessible"))
        return 2
    elseif ((MGI(x,y,"Coude bas") == "inaccessible")&& (MGI(x,y,"Coude haut") != "inaccessible"))
        return 3;
    else
        return 0
    end
end
```

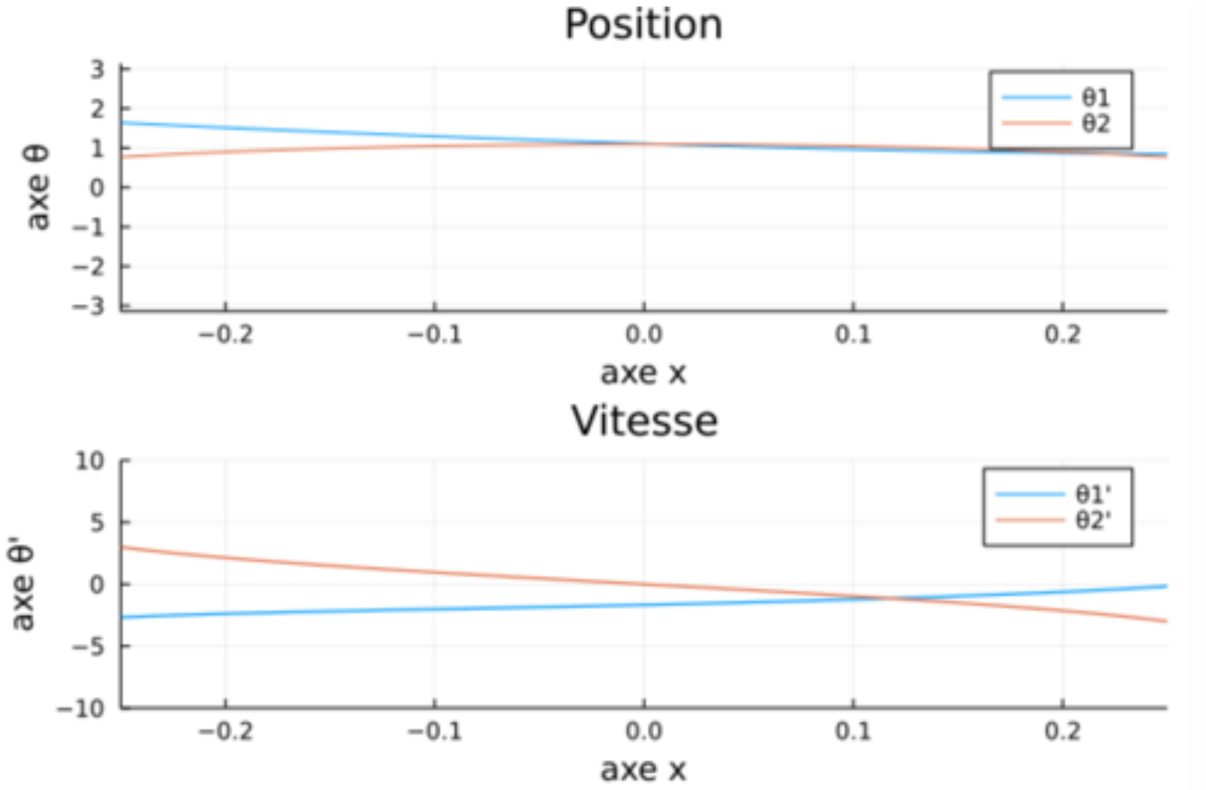
Sébastien Doyez
Lucien Reyser
MEA3
2022

```
function trace_angulaire_mci(R, x_dep, x_arr, y_arr)
    taille =0;
    pas = 1e-3;
    for i in x_dep:pas:x_arr
        taille = taille+1;
    end
    vec_θ1 = zeros(taille);
    vec_θ2 = zeros(taille);
    vec_speed_θ1 = zeros(taille);
    vec_speed_θ2 = zeros(taille);

    indice =1;
    X = zeros(taille);
    # remplisage des matrices
    for x in x_dep: pas: x_arr
        #vérification que x, y est atteignable :
        #X=[x;y_arr];
        OK= possible(R,x,y_arr);
        if (OK == 1)
            coude ="Coude bas";
        elseif (OK == 2)
            coude = "Coude bas";
        elseif (OK == 3)
            coude = "Coude  haut";
        end
        if (OK != 0)
            mgi=MGI(x,y_arr,coude);
            vec_θ1[indice] =mgi[1];
            vec_θ2[indice] =mgi[2];
            mci = MCI(mgi[1],mgi[2]);
            vec_speed_θ1[indice] = mci[1];
            vec_speed_θ2[indice] = mci[2];
            X[indice] = x;
            indice = indice +1;

        else
            print("sorti de l'espace disponible...");
            return 0;
        end
    end
    vec_θ1 = vec_θ1[1:indice-1];
    vec_θ2=  vec_θ2[1:indice-1];
    vec_speed_θ1 = vec_speed_θ1[1:indice-1];
    vec_speed_θ2 = vec_speed_θ2[1:indice-1];
    X=X[1:indice-1];
    figure6=plot(X,[vec_θ1,vec_θ2],title="Position",xlabel="axe
x",ylabel="axe θ",xlim=(x_dep,x_arr),ylim=(-pi,pi));
    figure7=plot(X,
[vec_speed_θ1,vec_speed_θ2],title="Vitesse",xlabel="axe x",ylabel="axe
θ'",linewidth=1.2,xlim=(x_dep,x_arr),ylim=(-10,10));
    fig8 = plot(figure6,figure7,layout=(2,1),label=["θ1" "θ2" "θ1'"
"θ2'"]);
    display(fig8)
    return 1;
end

trace_angulaire_mci(R, -0.25, 0.25, 0.8);
```

Une fois sorti de la boucle, il va falloir tracer sur deux figures différentes, la position et la vitesse : on a choisi également de retirer de nos matrice la dernière position et la dernière vitesse, car celle-ci ne présente plus d'intérêt, nous sommes à la position d'arrivée attendue.

Sébastien Doyez
Lucien Reyser
MEA3
2022

Ainsi, nous obtenons pour un test quelconque tel que x de départ est à -0.25, le x d'arrivée est à 0.25 et enfin le y d'arrivée est à 0.8 le tracé ci-dessous :



3-Commande dans l'espace de la tâche en utilisant le MGI

3.1 Contrôle du robot

- ***Représenter graphiquement par schéma bloc, le schéma de commande en position avec le MGI***

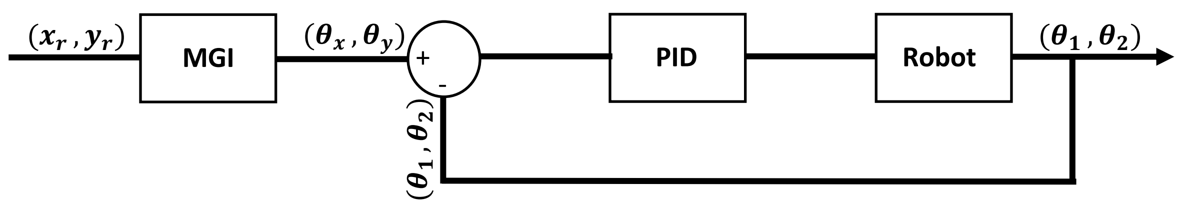


Schéma de commande en position avec le MGI

Sébastien Doyez
Lucien Reyser
MEA3
2022

- ***Réaliser la commande Julia+Vrep permettant de contrôler la position dans l'espace de la tâche***

Nous allons nous intéresser à la simulation de notre bras manipulateur sur CoppeliaSim. Dans les 3 parties précédentes, nous avions créé les fonctions MGD, MGI, MCD et MCI, qui vont nous être utile afin de faire fonctionner notre robot. Pour réaliser la commande permettant de contrôler la position dans l'espace de la tâche, nous allons utiliser la MGI. Nous allons donc procéder de la manière suivante :

```
A = possible(R,xdep,ydep);
if (A == 0 )
    print("départ impossible")
    return 0
elseif (A==1)
    word1="Coude haut"
    # peu importe car on peut prendre CH ou CB
elseif (A==2)
    word1 = "Coude bas";
elseif (A == 3)
    word1 = "Coude haut";
end
B = possible(R,xarr,yarr);
if (B == 0 )
    print("Position d'arrivée impossible")
    return 0
elseif (B==1)
    word2="Coude haut"
    # peu importe car on peut prendre CH ou CB
elseif (B==2)
    word2 = "Coude bas";
elseif (B == 3)
    word2 = "Coude haut";
end
```

Notre fonction MGI a besoin de 3 arguments, la coordonnée x et y de départ et la configuration du robot : « coude haut » ou « coude bas » … Il faut donc faire un test afin de savoir quelle configuration est la plus adapté. La fonction « possible » renvoie 0 si la position n'est pas accessible, 1 si les deux configurations sont possibles, 2 si celle-ci est accessible en coude bas ou enfin 3 si elle l'est en coude haut. Par défaut, nous avons choisi de prendre « coude haut » si les deux configurations sont possibles.

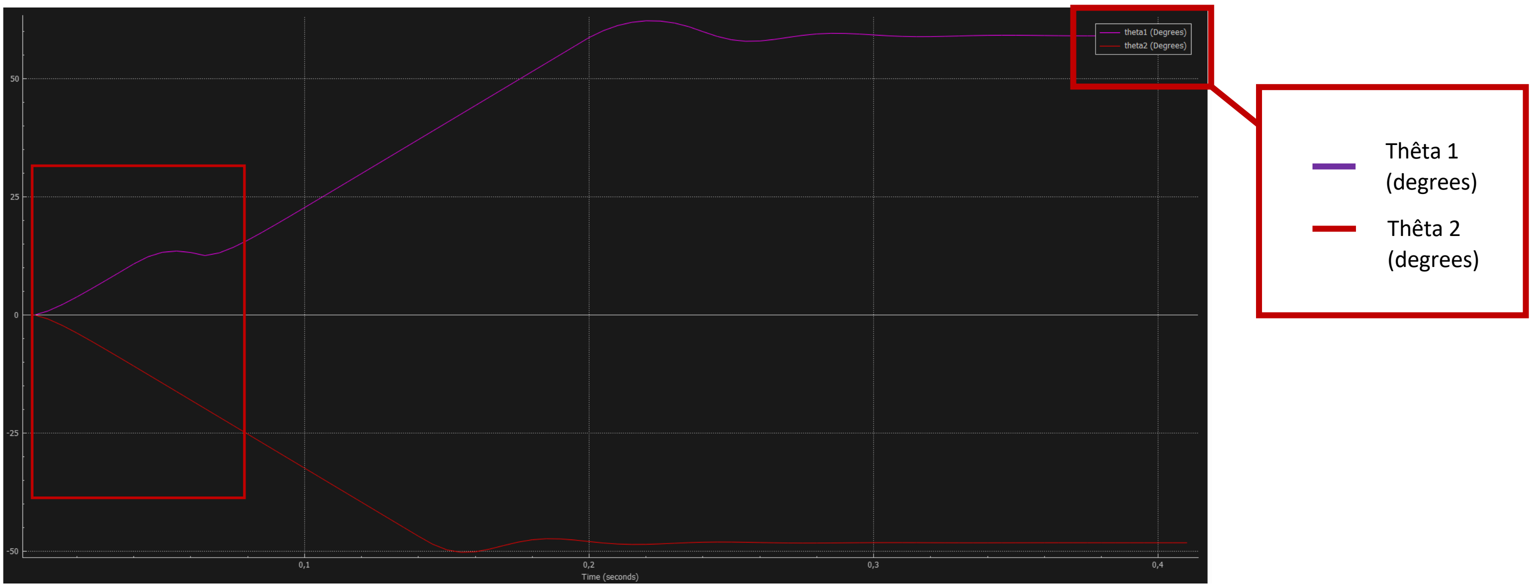
Maintenant que nous savons quelle configuration utilisée, nous pouvons faire appel à le MGI. Nous avons également mis en place une sécurité : si la position est inaccessible en coude bas et haut : on renvoie 0, ce qui a pour impact de nous faire sortir de la fonction.

Nous utilisons le MGI pour obtenir les angles ϑ1 et ϑ2. La commande « setjointposition » va alors nous permettre d'atteindre les angles calculés précédemment. Maintenant, il faut faire une boucle « while » avec un indice « t » qui sera décrémenter à chaque tour de boucle, ce qui nous permettra d'assurer une « sécurité » afin d'éviter tout problème de dépassement de temps. Nous allons utiliser une seconde fois la MGI afin d'obtenir les angles ϑ1_final et ϑ2_final … On réutilisera également la fonction « setjointposition » pour atteindre la position finale.

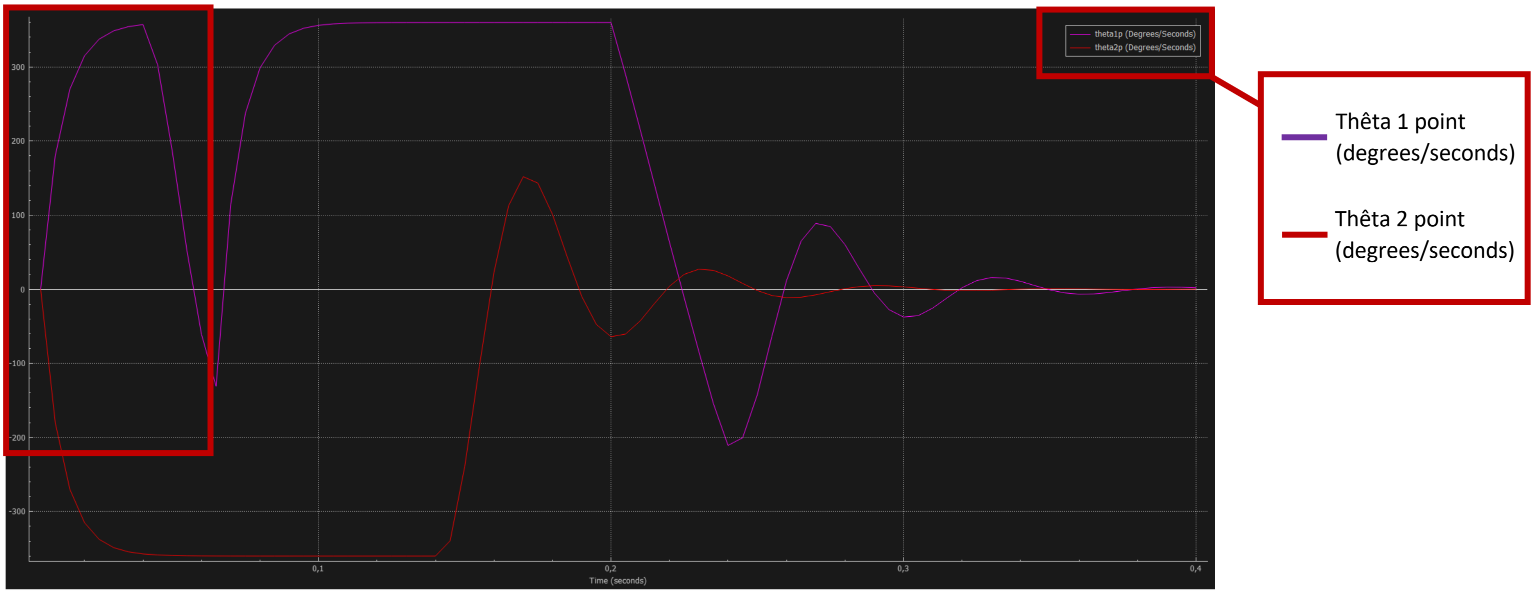
CoppeliaSim nous permet de visualiser le bras manipulateur, ainsi que la courbe de la position angulaire et la vitesse angulaire.

Pour un départ à la coordonnée : $P_{init1} = \binom{0.5}{0}$ *et une arrivée à la coordonnée* $P_{f1} = \binom{0.5}{0.6}$*, on obtient les deux courbes ci-dessous :*

Sébastien Doyez
Lucien Reyser
MEA3
2022



Courbe 1 : Position angulaire en fonction du temps



Courbe 2 : Vitesse angulaire en fonction du temps

Nous pouvons remarquer que la fonction « setjointposition » nous fait atteindre une position en faisant bouger le bras robotisé. La premier partie (encadré en rouge) sur le graphes représentant la position angulaire en fonction du temps (Courbe 1) et le graphe représentant la vitesse angulaire en fonction du temps (Courbe 2) ne sont pas à prendre en compte, il s'agit du déplacement de la position par défaut du bras à $P_{init1} = \binom{0.5}{0}$ … Maintenant si on s'intéresse à la courbes 2 on remarque que pour $\vartheta 1'$, la vitesse angulaire augmente assez rapidement à la manière d'un circuit RC en électronique, puis stagne à la vitesse de 360°/s. Enfin, rediminue lorsque le bras s'approche de la *position final, causant ainsi une oscillation amortie. De même pour ϑ2', on va retrouver un comportement identique, à l'exception que celui-ci inversé par rapport à $\vartheta 1'$.Ceci est dû au fait qu'il tourne dans le sens opposé. Maintenant si on s'intéresse à ϑ1 et ϑ2, on remarque alors que leur évolution est assez linéaire. On a tout de même un dépassement lorsque le bras arrive à la position finale.*

Maintenant intéressons-nous à l'étude automatique du système. Pour la vitesse angulaire, nous retrouvons pour ϑ'1 et ϑ'2 des dépassements de 50% ce qui peut être assez dangereux si le bras est utilisé dans le domaine de la médecine. Cependant cette oscillation peut être corriger via des correcteurs. Cette oscillation est à l'origine d'une oscillation sur ϑ1 et ϑ2 : nous avons un dépassement de 10°/s ce qui correspond à 16% ce n'est malheureusement pas négligeable.

En ce qui concerne la simulation pour la coordonnée $P_{init2} = \binom{-0.1}{0.5}$ et la coordonnée d'arrivée $P_{f2} = \binom{-0.6}{0.2}$, du fait que la position initiale soit inaccessible pour le robot, la simulation n'est pas réalisable

```
julia> testsimu2(-0.1,0.5,-0.6,0.2)
position de départ impossible0
```

4- Commande dans l'espace de la tâche en utilisant le MCI

4.1 Contrôle du robot

- ***Représenter graphiquement par schéma bloc, le schéma de commande en position avec le MCI***

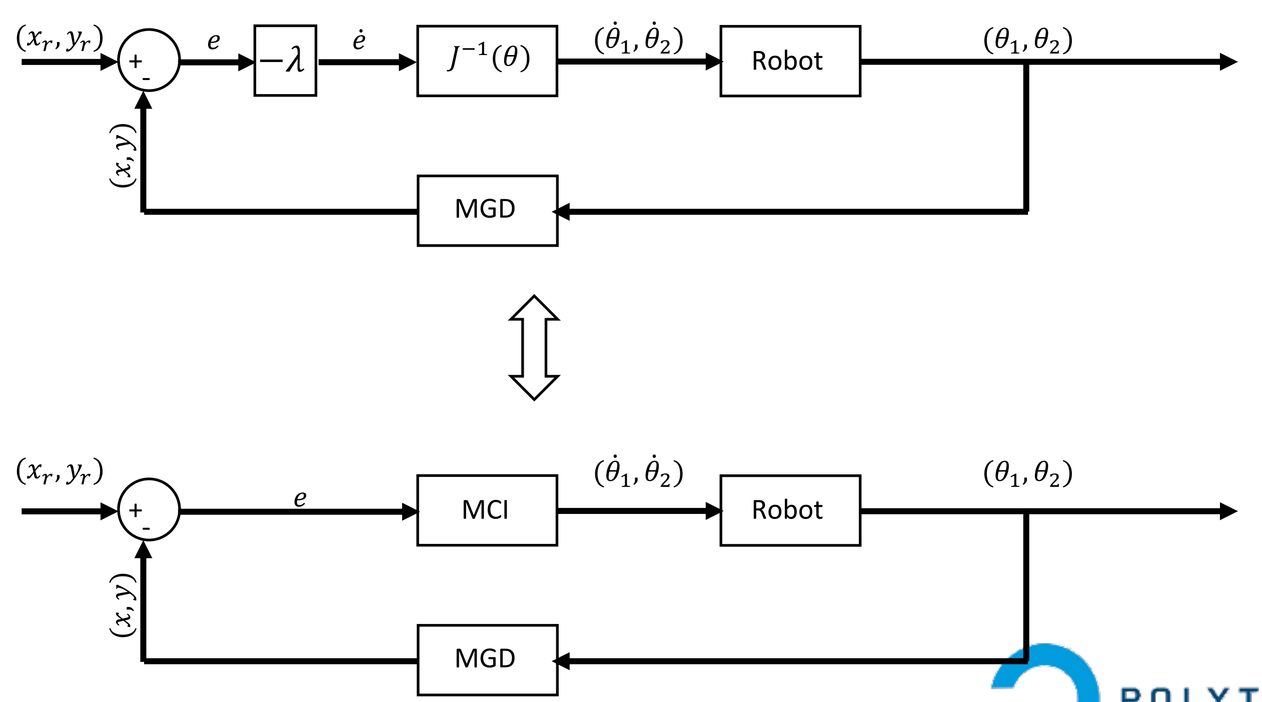


Schéma de commande en position avec le MCI

Sébastien Doyez
Lucien Reyser
MEA3
2022

- ***Réaliser la commande Julia+Vrep permettant de contrôler la position dans l'espace de la tâche***

Cette partie ressemble à la partie précédente, à l'exception que nous allons utiliser une commande vitesse : on utilisera alors la MCI.

On réalise les mêmes tests que décrit dans la partie « Commande dans l'espace de la tâche en utilisant la MGI » avec la fonction « possible » afin d'utiliser le MGI pour placer notre robot à la position initiale. Puis nous utilisons le MCI dans la même boucle « while » utilisé dans la partie 3.1. Celle-ci va nous fournir la vitesse angulaire.

Nous avons utilisé dans ce code (Figure 1) la commande cinématique vu en cours (Figure 2) :

```
if (t>0.05)
    ## maintenant on utilise la commande
    P = MGD(pos[1], pos[2]);
    Arr = [xa;ya];
    e = P- Arr
    if (commande_MCI(R,θ[1],θ[2],e,λ) ==0 )
        return 0;
    end
    θ_dev = commande_MCI(R,θ[1],θ[2],e,λ);

    qp=[θ_dev[1],θ_dev[2]]
```

Figure 1 : Extrait du code

Commande Cinématique

On souhaite contrôler la position d'un robot manipulateur en utilisant le MCI. Soit :

$$\dot{\theta} = J^{-1}\dot{e} \tag{12}$$

On choisit $e = X - X_d$. Avec X_d le vecteur position désiré de l'effecteur. On défini le comportement dynamique du robot avec la relation suivante

$$\dot{e} + \lambda e = 0$$

Figure 2 : Extrait du cours de Robotique

« e » est défini par la position actuelle (qui est définit dans notre code via le MGD) et « Arr » est la position désirée, autrement dit la position d'arrivée .

La fonction « commande_mci » est défini de la sorte (Figure 3) :

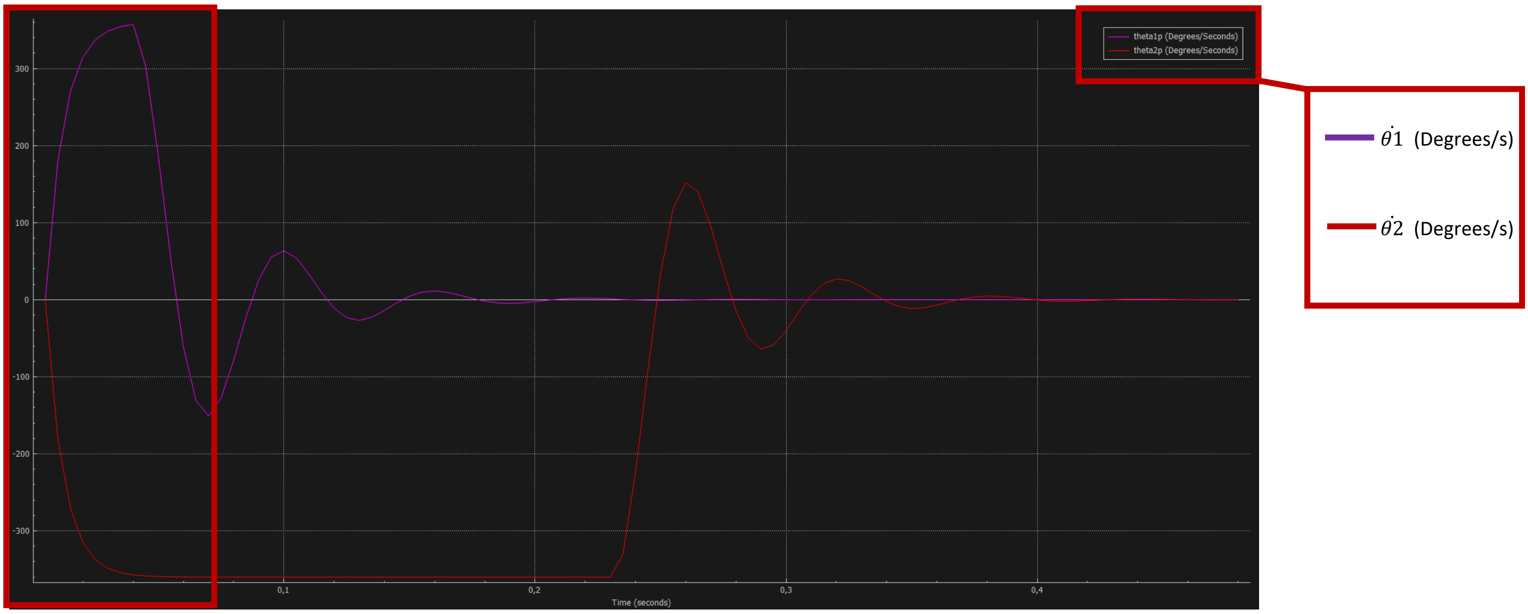
On calcule la matrice Jacobienne, (avec un test de déterminant car si le déterminant de J est nul, alors par définition, la matrice n'est pas inversible, et il sera inutile de poursuivre les calculs), la commande est alors définie par $\dot{\vartheta} = -\lambda J^{-1}(\theta)e$.

```
function commande_MCI(R,θ1,θ2,e,λ)
    J = Jacob_MCD(θ1,θ2);
    if (det(J) != 0)
        return(-λ * inv(J) * e);
    end
    print("J non inversible.")
    return 0;
end
```

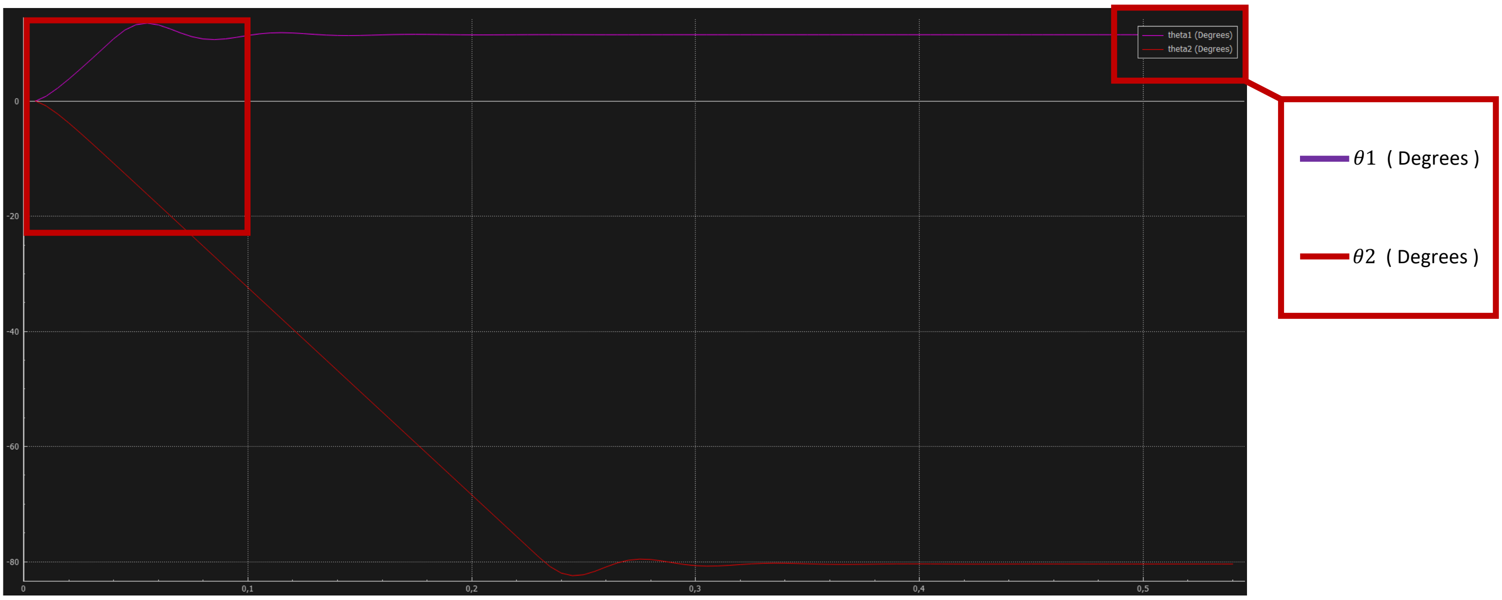
Figure 3 : fonction commande_MCI

Sébastien Doyez
Lucien Reyser
MEA3
2022

Intéressons-nous maintenant aux positions et vitesse angulaires donné par CoppeliaSim :



Courbe 1 : Vitesse angulaire en fonction du temps



Courbe 2 : Position angulaire en fonction du temps

Comme dis dans la partie 3.1, la premier partie (encadré en rouge) sur le graphes représentant la position angulaire en fonction du temps (Courbe 1) et le graphe représentant la vitesse angulaire en fonction du temps (Courbe 2) ne sont pas à prendre en compte, il s'agit du déplacement de la position par défaut du bras à la position initial.

Si on s'intéresse à la seconde partie des deux courbes, pour θ'1, on remarque que la vitesse angulaire diminue assez rapidement, à la manière d'un circuit RC en électronique, puis stagne à la vitesse de

-360°/s, et enfin augmente lorsque le bras s'approche de la position final, causant ainsi une oscillation amortie. De même pour θ2' qui ne va tourner que légèrement dans l'autre sens, on retrouve une oscillation amortie lorsque la position de θ2 est atteinte. Maintenant si on s'intéresse à θ1 et θ2, on remarque que leur évolution est assez linéaire, mais on a tout de même un dépassement lorsque le bras est arrivé à la position finale.

D'un point de vue d'automaticien, nous pouvons remarquer que les dépassements en vitesse sont de l'ordre de 150° /s pour θ2' (soit presque 50%, ce qui est assez important, dangereux si on a besoin d'une certaine précision (dans le domaine médicale par exemple) et améliorable via des correcteurs) . Le dépassement de θ1' est quant à lui de 50 °/s (car cette vitesse angulaire augmente puis diminue très rapidement ce qui fait que la valeur finale n'est pas 360°/s comme dans la première partie, mais 0°/s). Tout comme à la partie 3.1, cette oscillation va causer une autre oscillation sur θ1 et θ2 : nous avons un dépassement d'environ 10°/s ce qui correspond à 16% ce qui n'est pas négligeable.

Maintenant, si nous comparons nos deux commandes : la commande en position permet d'atteindre la position désirée plus rapidement, mais demande au bras de faire plus de mouvement.