EXPLICATION DU PROGRAMME

Dans le cadre de notre projet qui consistait à appliquer la cinématique inverse au robot Poppy Torso dans un jeu de morpion à l'aide du simulateur V-REP nous avons développé notre programme de jeu en langage Python dont le code principal se trouve dans le script nommé « kinematic_torso.py ».

1. Description du script « kinematic_torso.py »

Ce script permettant le déroulement complet du jeu est organisé, dans l'ordre d'exécution, en plusieurs parties.

1.1. Encodage et importation de modules

Le code relatif à l'encodage et à l'importation de modules se présente comme suit :

```
# -*-coding: UTF-8 -*

# IMPORTATION DE MODULES
import time
from functions_game import*
from pypot.vrep import from_vrep
from pypot.creatures import PoppyTorso
```

La première ligne de code correspond à l'encodage UTF-8 pour assurer la prise en compte des caractères accentués. Après vient l'importation des modules que nous avons utilisés.

- Le module « time » pour pouvoir manipuler le temps dans notre programme.
- Le module « functions_game » qui contient l'ensemble des fonctions que nous avons développé. L'explication de son contenu fera l'objet du point 2.
- Le module « from_vrep » pour permettre la création du robot sous V-REP.
- Le module « PoppyTorso » pour la création d'un objet de type PoppyTorso.

Notons à ce niveau que l'installation de la librairie IKPy entraîne l'intégration du script « kinematics .py » dans la librairie pypot (C:\ProgramData\Anaconda2\Lib\site-packages\pypot). Ce script contient les modules de calculs de la cinématique directe et inverse.

1.2. Création des objets du jeu de morpion

Après l'encodage et l'importation des modules nous avons la création des différents objets du jeu. Le code pour la création de ces objets se présente comme suit :

```
# CREATION DU ROBOT POPPY TORSO
poppy =PoppyTorso(simulator='vrep')

# POSITION INITIALE DU ROBOT
motor_angles_init = [-2,0,-20,-5,-15,10,-10,20,0,0,0,15,-10]
move_time = 1.5
move_robot(poppy,motor_angles_init,move_time)

# CREATION DU PION ET DES CASES
scene_morpion(poppy)
```

La première ligne de code permet de créer un objet « poppy » de type PoppyTorso et d'établir sa connexion avec le simulateur V-REP.

Ensuite nous avons les variables « motor_angles_init » et « move_time » qui contiennent respectivement la position angulaire initiale des treize moteurs et la durée pour le positionnement de chaque moteur. Juste après nous avons l'appel de la fonction « move_robot () » qui permet de positionner les angles moteurs du robot Poppy Torso dans la configuration définie dans « motor_angles_init ». Le code de cette fonction provient du projet « Main préhensile pour le robot Poppy et interface Tangible » et se trouve dans le module « functions_game ».

La dernière ligne de code est consacrée à la création du pion et des neuf (09) cases de la grille de jeu. Cette création se fait par l'appel de la fonction « scene_morpion () » dont le code se trouve également dans le module « functions_game ».

1.3. Déroulement du jeu

Le démarrage d'une partie de jeu est la phase qui vient juste après la création des objets. Le déroulement du jeu peut être scindé en deux (02) parties : la communication des ordres par l'utilisateur et l'exécution des ordres par le robot. Les lignes de codes ci-après décrivent le déroulement du jeu.

❖ Communication des ordres par l'utilisateur

Après la mise en place de l'espace de jeu l'utilisateur est invité à saisir le nombre de pions à jouer par le robot. La valeur saisie par l'utilisateur est stockée dans la variable « nb_pion ».

Une fois le nombre de pions renseigné, nous entrons dans une boucle « while » dans laquelle nous ne sortirons lorsque le robot aura joué le nombre de pions saisi précédemment. A ce niveau nous avons défini deux (02) variables :

- « nb_tour » un compteur de type entier qui contiendra le nombre de pions joué.
- « play » de type booléen qui nous permettra de sortir de la boucle « while » à condition que la valeur de « nb_tour » soit égale à la valeur de « nb_pion ».

```
# DEMANDE DU NOMBRE DE PIONS A JOUER A L'UTILISATEUR
print("================================================ \n")
nb_pion = input("SAISISSEZ LE NOMBRE DE PIONS A JOUER : ")

play=True
nb_tour = 0
while (play):

    if (nb_tour == nb_pion):
        play=False
    else:
        # AFFICHAGE DU NOMBRE DE TOURS
        print("================================================ \n")
        nb_tour = nb_tour + 1
        print("POPPY TORSO DOIT JOUER LE PION " ,nb_tour)

        # DEMANDE DE LA CASE A ATTEINDRE A L'UTILISATEUR
        row = input("INDIQUEZ LA LIGNE DE LA CASE A ATTEINDRE (0,1 ou 2):\n")
        col = input("INDIQUEZ LA COLONNE DE LA CASE A ATTEINDRE(0,1 ou 2):\n")

        if(row ==0):
            if(col == 0):
                xcase,ycase,zcase= (-0.10,-0.25,0.03)
            elif(col == 1):
                xcase,ycase,zcase= (0.00,-0.25,0.03)
            elif(col == 2):
                xcase,ycase,zcase= (0.10,-0.25,0.03)
        elif(row == 1):
            if(col == 0):
                xcase,ycase,zcase= (-0.10,-0.20,0.03)
            elif(col == 1):
                xcase,ycase,zcase= (0.00,-0.20,0.03)
            elif(col == 2):
                xcase,ycase,zcase= (0.10,-0.20,0.03)
        elif(row == 2):
            if(col == 0):
                xcase,ycase,zcase= (-0.10,-0.15,0.03)
            elif(col == 1):
                xcase,ycase,zcase= (0.00,-0.15,0.03)
            elif(col == 2):
                xcase,ycase,zcase= (0.10,-0.15,0.03)

        # EXECUTION DE LA DEMANDE PAR LE ROBOT : CINEMATIQUE INVERSE
        move_kinematic(poppy,xcase,ycase,zcase)
        # POSITION INITIALE DU ROBOT
        move_robot(poppy,motor_angles_init,move_time)

# FIN DE LA PARTIE DE JEU
print("============= FIN DE LA PARTIE DE JEU =============")
time.sleep(move_time)
poppy.stop_simulation()
```

A chaque tour de jeu, l'utilisation est invité à indiquer la ligne puis la colonne de la case que le robot doit atteindre avec sa main droite. Les valeurs saisies sont stockées respectivement dans les variables « row » et « col ».

En fonction de la case ainsi désignée par ces ligne et colonne, les coordonnées cartésiennes correspondantes sont affectées aux variables « xcase », « ycase » et « zcase ».

Nous pouvons remarquer que les valeurs des coordonnées correspondent aux valeurs utilisées pour la création des cases à l'exception des valeurs de zcase. En effet, pour la création du pion et des cases le niveau zéro (0) de la côte (l'axe z) est pris au niveau du sol. Par contre, pour le robot Poppy Torso le niveau zéro (0) de la côte correspond au-dessus de la table. Nous avons donc fixé la valeur de « zcase » à 3 cm.

❖ **Exécution des ordres par le robot**

Une fois que les coordonnées cartésiennes de la case à atteindre sont affectées aux variables « xcase », « ycase » et « zcase » le robot peut maintenant se mouvoir. Pour cela nous avons l'appel de la fonction « move_kinematic () » qui permet au robot Poppy Torso, en utilisant la cinématique inverse, d'effectuer avec sa main droite :

- une trajectoire en cloche pour se positionner juste au-dessus du pion.
- une trajectoire en cloche pour se positionner juste au-dessus de la case indiquée par l'utilisateur.
- une trajectoire rectiligne pour se retourner à la position de départ.

Le code de cette fonction se trouve également dans le module « functions_game ».

Juste après nous avons l'appel de la fonction « move_robot () » pour permettre aux treize (13) moteurs du robot Poppy Torso d'être dans la configuration initiale.

2. Description du module « functions_game »

La première ligne de code de ce module correspond à l'encodage UTF-8. Juste après nous avons l'importation des modules « time » pour la manipulation du temps dans notre programme et « math » pour l'utilisation des fonctions mathématiques.

2.1. Fonction « scene_morpion () »

La première fonction définie dans le module est la fonction « scene_morpion () ». Cette fonction chargée de la création du pion et des neuf (09) cases de la grille de jeu prend comme paramètre d'entrée l'objet « poppy ».

```
def scene_morpion(poppy):
    # CREATION DE CASES ET PIONS
    objet = poppy._controllers[0].io

    name_pion = 'pion'
    pos_pion = [-0.30,-0.15,0.75] # X, Y, Z
    sizes_pion = [0.03, 0.03,0.08] # in meters
    mass_pion = 2 #  in kg
    objet.add_cube(name_pion, pos_pion, sizes_pion, mass_pion)

    name_case = 'case1'
    pos_case = [-0.10,-0.25,0.75] # X, Y, Z (0.0)
    sizes_case = [0.08, 0.04,0.001] # in meters (x,y,h)
    mass_case = 2 # in kg
    objet.add_cube(name_case, pos_case, sizes_case, mass_case)
    name_case = 'case2'
    pos_case = [0.0,-0.25,0.75] # X, Y, Z (0.1)
    sizes_case = [0.08, 0.04,0.001] # in meters (x,y,h)
    mass_case = 2 # in kg
    objet.add_cube(name_case, pos_case, sizes_case, mass_case)
    name_case = 'case3'
    pos_case = [0.10,-0.25,0.75] # X, Y, Z (0.2)
    sizes_case = [0.08, 0.04,0.001] # in meters (x,y,h)
    mass_case = 0.05 # in kg
    objet.add_cube(name_case, pos_case, sizes_case, mass_case)

    name_case = 'case4'
    pos_case = [-0.10,-0.20,0.75] # X, Y, Z (1.0)
    sizes_case = [0.08, 0.04,0.001] # in meters (x,y,h)
    mass_case = 2 # in kg
    objet.add_cube(name_case, pos_case, sizes_case, mass_case)
    name_case = 'case5'
    pos_case = [0.0,-0.20,0.75] # X, Y, Z (1.1)
    sizes_case = [0.08, 0.04,0.001] # in meters (x,y,h)
    mass_case = 2 # in kg
    objet.add_cube(name_case, pos_case, sizes_case, mass_case)
    name_case = 'case6'
    pos_case = [0.10,-0.20,0.75] # X, Y, Z (1.2)
    sizes_case = [0.08, 0.04,0.001] # in meters (x,y,h)
    mass_case = 2 # in kg
    objet.add_cube(name_case, pos_case, sizes_case, mass_case)

    name_case = 'case7'
    pos_case = [-0.10,-0.15,0.75] # X, Y, Z (2.0)
    sizes_case = [0.08, 0.04,0.001] # in meters (x,y,h)
    mass_case = 2 # in kg
    objet.add_cube(name_case, pos_case, sizes_case, mass_case)
    name_case = 'case8'
    pos_case = [0.0,-0.15,0.75] # X, Y, Z (2.1)
    sizes_case = [0.08, 0.04,0.001] # in meters (x,y,h)
    mass_case = 2 # in kg
    objet.add_cube(name_case, pos_case, sizes_case, mass_case)
    name_case = 'case9'
    pos_case = [0.10,-0.15,0.75] # X, Y, Z (2.2)
    sizes_case = [0.08, 0.04,0.001] # in meters (x,y,h)
    mass_case = 2 # in kg
    objet.add_cube(name_case, pos_case, sizes_case, mass_case)
```

Son contenu commence par la création d'un objet « objet_io » instancié à l'interface io du robot afin de pouvoir faire apparaître nos cases et pion sur la table de Poppy Torso du simulateur robotique V-REP.

Nous avons par la suite utilisé la fonction « objet_io.add_cube() » pour créer le pion et les neuf (09) cases. Cette fonction prend en paramètre le nom, la position, les dimensions et la masse de l'objet à créer.

2.2. Fonction « move_kinematic () »

La second fonction de ce module est la fonction « move_kinematic () ». C'est cette fonction qui, à l'aide de la cinématique inverse, permet au robot Poppy Torso d'effectuer avec sa main droite un mouvement en cloche pour se positionner dans un premier temps juste au-dessus du pion, ensuite un mouvement en cloche pour se positionner juste au-dessus de la case indiquée par l'utilisateur puis un mouvement rectiligne pour ramener sa main droite à la position de départ.
Elle a comme paramètres d'entrée l'objet « poppy » et les coordonnées cartésiennes de la case choisie par l'utilisateur.

```
def move_kinematic(poppy,xcase,ycase,zcase):
    # MOUVEMENT DU ROBOT PAR L'APPLICATION DE LA CINEMATIQUE INVERSE
    x_main_init,y_main_init,z_main_init = (-0.19,-0.15,0.10)
    x_main,y_main,z_main = (x_main_init,y_main_init,z_main_init)
    x_pion,y_pion,z_pion,z_pion_max = (-0.30,-0.15,0.08,0.10)
    x_limit_rot = 0.06
    m_time = 0.0001
    pause = 2.5
    pas = 0.005
    pi = 3.14
    g = 10

# Mouvement 1 : Prendre le pion
    t = 0
    a = 2*pi/3
    V1 = calcul_parametre(a,x_main_init,z_main_init,x_pion,z_pion_max)
    k = (y_main_init - y_pion)/(x_main_init - x_pion)
    V2 = k*V1*cos(a)
    while (x_main > x_pion):
        t = t + pas
        x_main = V1*cos(a)*t + x_main_init
        y_main = V2*t + y_main_init
        z_main = -0.5*g*(t**2) + V1*sin(a)*t + z_main_init
        poppy.r_arm_chain.goto((x_main,y_main,z_main),m_time,wait=True)
    time.sleep(pause)

# Mouvement 2 : Jouer le pion
    t = 0
    a = pi/4
    V1 = calcul_parametre(a,x_pion,z_pion,xcase,zcase)
    k = (ycase - y_pion)/(xcase - x_pion)
    V2 = k*V1*cos(a)
    while (x_main < xcase):
        t = t + pas
        x_main = V1*cos(a)*t + x_pion
        y_main = V2*t + y_pion
        z_main = -0.5*g*(t**2) + V1*sin(a)*t + z_pion_max
        poppy.r_arm_chain.goto((x_main,y_main,z_main),m_time,wait=True)
        # contrôle de l'angle de rotation du moteur abs_z
        if (x_main > x_limit_rot):
            poppy.r_arm_chain.active_links_mask[1]=True
    time.sleep(pause)

# Mouvement 3 : Retour à la position initiale
    # deplacement selon z
    while(z_main < z_main_init):
        z_main = z_main + pas
        poppy.r_arm_chain.goto((x_main,y_main,z_main),m_time,wait=True)
    # deplacement dans le plan (O,x,y)
    k = (y_main_init - ycase)/(x_main_init - xcase)
    while (x_main > x_main_init):
        x_main = x_main - pas
        y_main = k*(x_main - xcase) + ycase
        poppy.r_arm_chain.goto((x_main,y_main,z_main),m_time,wait=True)
        # contrôle de l'angle de rotation du moteur abs_z
        if(x_main < x_limit_rot):
            poppy.r_arm_chain.active_links_mask[1] = False
    time.sleep(pause)
```

❖ Définition des variables

Son contenu commence par la définition des variables :

- « x_main_init », « y_main_init » et « z_main_init » contiennent les coordonnées cartésiennes de la position initiale de la main droite du robot.
- « x_main », « y_main », et « z_main » contiennent les coordonnées cartésiennes de la position courante de la main droite du robot.
- « x_pion », « y_pion » et « z_pion_max » contiennent les coordonnées cartésiennes pour être juste au-dessus du pion.
- « z_pion » contient la hauteur du pion.
- « x_limit_rot » contient la valeur en abscisse pour laquelle le moteur « abs_z » devient actif dans le but d'atteindre certaines cases par rotation.
- « m_time » contient le temps à mettre par la main droite du robot pour passer d'une position élémentaire à une autre.
- « pause » contient le temps de pause entre chaque mouvement.
- « pas » contient le pas de déplacement.
- « pi » contient la valeur de π.
- « g » contient la valeur de l'intensité de la pesanteur.

❖ Mouvement 1 : prendre le pion

Après la définition des variables vient les lignes de codes permettant au robot de positionner sa main droite juste au-dessus du pion en effectuant un mouvement en cloche. Les équations horaires décrivant le mouvement sont :

$X(t) = V_1.\cos(a).t + Xi$
$Y(t) = V_2.t + Yi$
$Z(t) = -1/2.g.t^2 + V_1.\sin(a).t + Zi$

Dans ce système d'équations X(t), Y(t) et Z(t) représentent la position courante de la main droite de Poppy Torso c'est-à-dire « x_main », « y_main », et « z_main ».
Les constantes Xi, Yi et Zi correspondent aux coordonnées cartésiennes de la position initiale de cette main. Il s'agit du contenu des variables « x_main_init », « y_main_init » et « z_main_init ».

Avant que le robot n'effectue le mouvement nous avons d'abord calculé les paramètres inconnus du système d'équations que sont V_1 et V_2 en fixant la valeur de l'angle « a » à 2π/3. La détermination de la valeur de la vitesse V_1 se fait par la fonction « calcul_parametre () ». Quant à la valeur de la vitesse V_2 elle se détermine par la relation $V_2 = k.V_1.\cos(a)$ avec k = (y_main_init - y_pion) / (x_main_init - x_pion).

Pour le pilotage du bras droit du robot Poppy Torso nous avons utilisé la fonction « poppy.r_arm_chain.goto () » de la librairie IKPy. Cette fonction applique la

cinématique inverse au bras droit de Poppy Torso. En effet, elle calcule et affecte la position angulaire et l'orientation qu'il faut, pour chacun des sept (07) moteurs [r_elbw_y ; r_arm_z ; r_shoulder_x ; r_shoulder_y ; bust_x ; bust_y ; abs_z] du bras droit de Poppy Torso, pour atteindre la position souhaitée. Par défaut seuls les quatre (04) moteurs [r_elbw_y ; r_arm_z ; r_shoulder_x ; r_shoulder_y] sont actifs. Elle prend comme paramètres d'entrée :

- les coordonnées cartésiennes de la position à atteindre.

Dans notre cas il s'agit du contenu des variables « x_main », « y_main » et « z_main ». Ces variables contiendront les coordonnées cartésiennes des différentes positions élémentaires renvoyées, à chaque pas, par le système d'équations horaires du mouvement.

- le temps pour atteindre la position souhaitée.

Ce temps est défini par la variable « m_time ».

- la mise en attente ou non de l'exécution de la suite du programme.

Nous avons opté pour une mise en attente à travers « wait = True ».

Le code de pilotage du bras droit du robot Poppy Torso se trouve dans une boucle « while » qui permet de parcourir la trajectoire définie par le système d'équations horaires évoqué précédemment.

❖ **Mouvement 2 : jouer le pion**

Après une pause de 2,5 secondes au-dessus du pion, le robot effectue une fois de plus avec sa main droite un mouvement en cloche pour la positionner juste au-dessus de la case indiquée par l'utilisateur.

Le pilotage de la main se fait toujours selon le même procédé. La différente se situe à quatre (04) niveaux :

- La valeur de l'angle « a » est cette fois fixée à π/4.
- L'expression du coefficient k = (ycase - y_pion) / (xcase - x_pion).
- Les constantes Xi, Yi et Zi du système d'équations correspondent ici aux coordonnées cartésiennes de la position de la main juste au-dessus du pion. Il s'agit du contenu des variables « x_pion », « y_pion » et « z_pion_max ».
- L'activation du moteur abs_z à travers la ligne de code « poppy.r_arm_chain.active_links_mask[1] = True » à condition que la valeur de « x_main » soit supérieure à la valeur de « x_limit_rot ». L'objectif est de permettre à la main du robot d'atteindre certaines cases par rotation.

❖ **Mouvement 3 : retour à la position initiale**

2,5 secondes après avoir joué le pion le robot Poppy Torso doit renvoyer sa main droite à la position de départ. Le robot effectue à ce niveau un mouvement rectiligne uniforme selon l'axe z pour éviter les pions déjà posés et un mouvement rectiligne uniforme dans le plan (O,x,y).

Pour le mouvement selon l'axe z, nous avons fait varier la valeur de « z_main » de « zcase » à « z_main_init » en fixant un pas. De manière concrète nous avons utilisé une boucle « while » dans laquelle pour chaque variation élémentaire de « z_main » nous avons fait appel à la fonction « poppy.r_arm_chain.goto () » pour piloter la main droite du robot à cette nouvelle position de « z_main », le contenu de « x_main » et « y_main » étant inchangé.

Une fois que la position de « z_main_init » est atteinte par « z_main », la main du robot effectue un mouvement rectiligne uniforme dans le plan (O,x,y). Les équations horaires décrivant le mouvement sont :

$X(t) = V_x.t + Xi$
$Y(t) = V_y.t + Yi$

Par la suite en exprimant Y(t) en fonction de X(t) nous obtenons $Y(X) = k(X-Xi) + Yi$ avec $k = (Yf - Yi) / (Xf - Xi)$.

Dans cette dernière relation X et Y représentent la position courant de la main droite de Poppy Torso dans le plan (O,x,y) c'est-à-dire « x_main » et « y_main ».
Les constantes Xi et Yi correspondent aux coordonnées cartésiennes dans le plan (O,x,y) de la case indiquée par l'utilisateur. Il s'agit du contenu des variables « xcase » et « ycase ».
Pour l'exécution du mouvement rectiligne dans le plan (O,x,y) nous avons encore utilisé une boucle « while » dans laquelle, pour chaque variation élémentaire de « x_main » de « xcase » à « x_main_init », nous avons calculé la nouvelle valeur de « y_main » puis fait appel à la fonction « poppy.r_arm_chain.goto () » pour piloter la main droite du robot à cette nouvelle position de « x_main » et « y_main », le contenu de « z_main » étant inchangé.

2.3. Fonction « calcul_parametre () »

La fonction « calcul_parametre () » est chargée de calculer la vitesse V_1 et la hauteur maximale h dans le cas d'un mouvement en cloche. Elle prend comme paramètre d'entrée l'angle « a » que fait la vitesse initiale V_1 avec l'axe des abscisses et les coordonnées cartésiennes des positions initiale et finale dans le plan (O,x,y) puis retourne la valeur les valeurs de V_1.
À partir des équations horaires de mouvement en cloche nous avons déterminé l'expression de la portée X_f qui vaut :

$$X_f = \frac{V_1}{g}\cos(a)\left[V_1 \sin(a) + \sqrt{(V_1 \sin(a))^2 + 2g(Z_i - Z_f)}\right] + X_i$$

Ensuite pour la valeur de l'angle « a » donnée en paramètre nous avons cherché la valeur de V_1 (en faisant varier V_1) qui donne une valeur de X_f proche de la valeur de X_f à atteindre.

Enfin, pour la valeur de V_1 retenue nous avons calculé la hauteur maximale h du mouvement en cloche en utilisant l'expression $h = \frac{1}{2g}(V_1 \sin(a))^2 + Z_i$

Le code de cette fonction se présente comme suit :

```
def calcul_parametre(a,Xi,Zi,Xf,Zf):
    # CALCULS DES PARAMETRES : VITESSE V1 ET HAUTEUR MAXIMALE h
    g = 10
    t = 0
    pi = 3.14
    V = 0
    pas = 0.0001
    NonTrouver = True
    while(NonTrouver):
        d = (V/g)*cos(a)*((V*sin(a))+sqrt(((V*sin(a))**2) + 2*g*(Zi-Zf))) + Xi
        if ((d >= Xf) and (Xf != -0.30)) :
            NonTrouver = False
        elif((abs(d) >= abs(Xf)) and (Xf ==-0.30)) :
            NonTrouver = False
        else :
            V = V + pas
    h = (0.5*(V**2)*((sin(a))**2))/g + Zi
    return(V)
```

2.4. Fonction « move_robot () »

Cette fonction permet de positionner les treize (13) moteurs du robot dans la configuration précisée dans la variable mortor_angles.

```
def move_robot(poppy,motor_angles,mov_time):
    # MOUVEMENT DU ROBOT PAR LA POSITION ANGULAIRE
    delay = 0.0001
    poppy.motors[0].goto_position(motor_angles[0],mov_time)
    time.sleep(delay)
    poppy.motors[1].goto_position(motor_angles[1],mov_time)
    time.sleep(delay)
    poppy.motors[2].goto_position(motor_angles[2],mov_time)
    time.sleep(delay)
    poppy.motors[3].goto_position(motor_angles[3],mov_time)
    time.sleep(delay)
    poppy.motors[4].goto_position(motor_angles[4],mov_time)
    time.sleep(delay)
    poppy.motors[5].goto_position(motor_angles[5],mov_time)
    time.sleep(delay)
    poppy.motors[6].goto_position(motor_angles[6],mov_time)
    time.sleep(delay)
    poppy.motors[7].goto_position(motor_angles[7],mov_time)
    time.sleep(delay)
    poppy.motors[8].goto_position(motor_angles[8],mov_time)
    time.sleep(delay)
    poppy.motors[9].goto_position(motor_angles[9],mov_time)
    time.sleep(delay)
    poppy.motors[10].goto_position(motor_angles[10],mov_time)
    time.sleep(delay)
    poppy.motors[11].goto_position(motor_angles[11],mov_time)
    time.sleep(delay)
    poppy.motors[12].goto_position(motor_angles[12],mov_time)
    time.sleep(delay)
```