CHOPIN Alexandre
HEBRAUD Julien
MAGRO Valentin
PORTE Matthieu

# Dossier technique – Robot piloté à distance

## ***I – Problématique et description :***

La problématique, à laquelle doit répondre le robot :

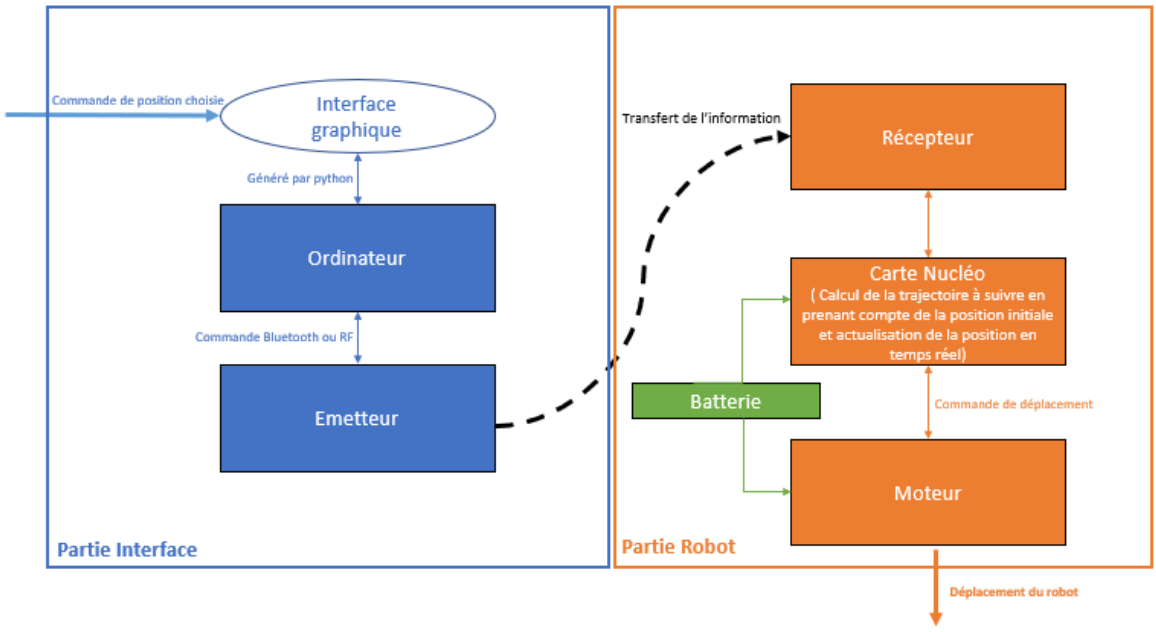
*Comment transmettre un ordre à un robot pour qu'il se déplace d'un point initial à un point final dans un milieu connu, à l'aide d'un ordinateur tout en respectant des contraintes de rapidité, de fiabilité et d'autonomie ?*

Cette problématique se sépare alors en deux points principaux :

- La conception d'une interface humain-machine afin de piloter le robot à distance, qui a été réalisée à l'aide de Python et du module Tkinter qui permet de construire des interfaces graphiques.
- La simulation du robot et le respect des contraintes, réalisée sous Matlab et Simulink.

A cause du télétravail, la réalisation du robot s'est réduite à la simulation du robot. Les contraintes de rapidité et de fiabilité ont été traité. Cependant, la contrainte d'autonomie n'a pas été traitée à cause des conditions de travail.

## ***II – Schéma complet de l'architecture :***



## ***III – Interface humain-machine :***

L'interface humain-machine a été réalisé sous Python et à l'aide du module Tkinter. Ce dernier permet la création d'interfaces graphiques complets. Il a pour but de servir de commande au robot pour l'utilisateur.

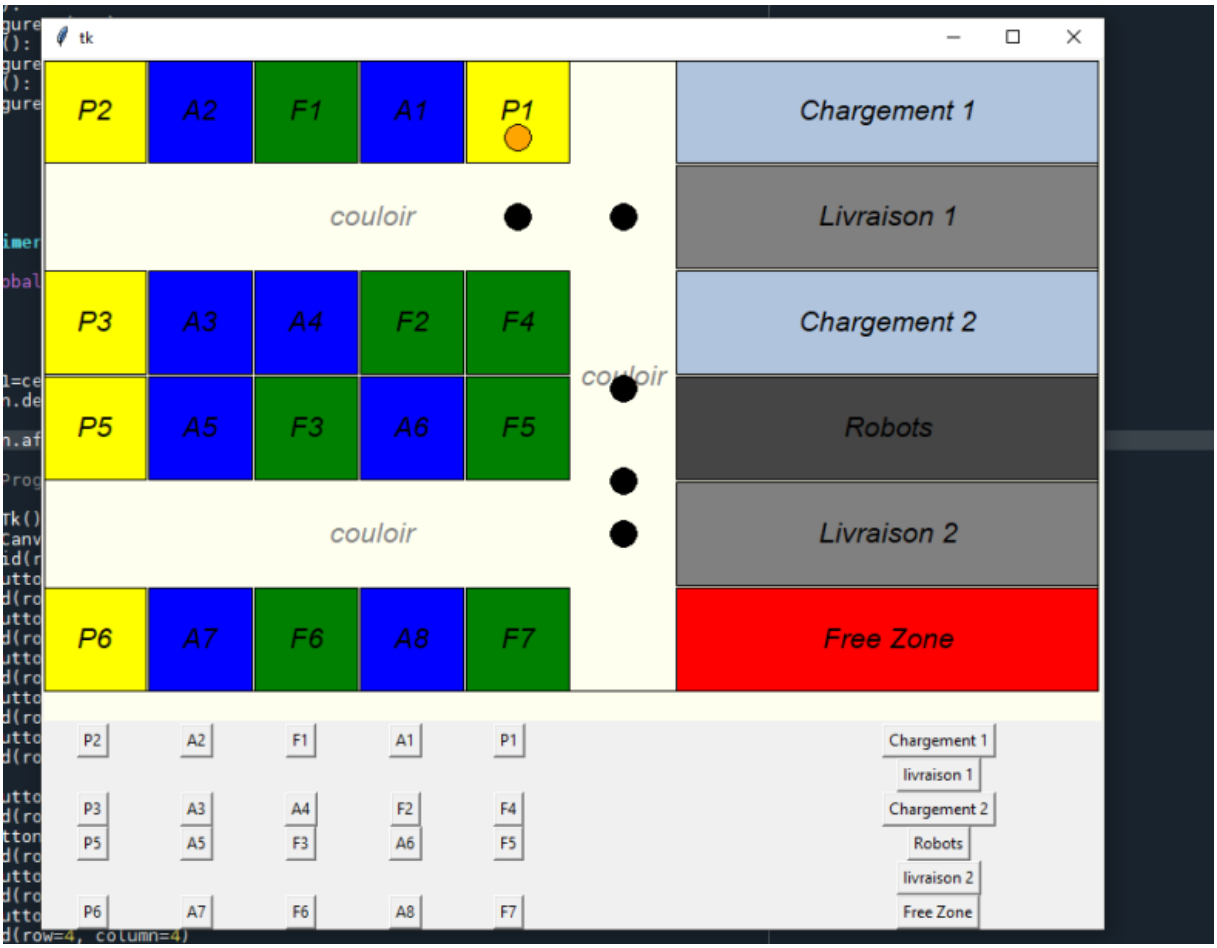



***Figure 1 – Plan de l'interface***

L'interface représente le plan de l'atelier fourni par l'entreprise dans lequel le robot doit se déplacer. Il permet alors à l'aide de boutons situés dans la section du bas de sélectionner l'endroit où l'on souhaite diriger le robot. Le point orange permet de marquer l'endroit de la dernière commande effectuée au robot. De plus l'interface affiche la trajectoire en temps réelle du robot par un disque noir depuis la dernière commande.
On utilise la rebrique canvas de tkinter pour dessiner des rectangles, les cercles et ajouter du texte. La figure canvas et les boutons sont placer dans la fenêtre avec l'outil grid .

L’algorithme principale fonctionne suivant le schéma ci-dessous

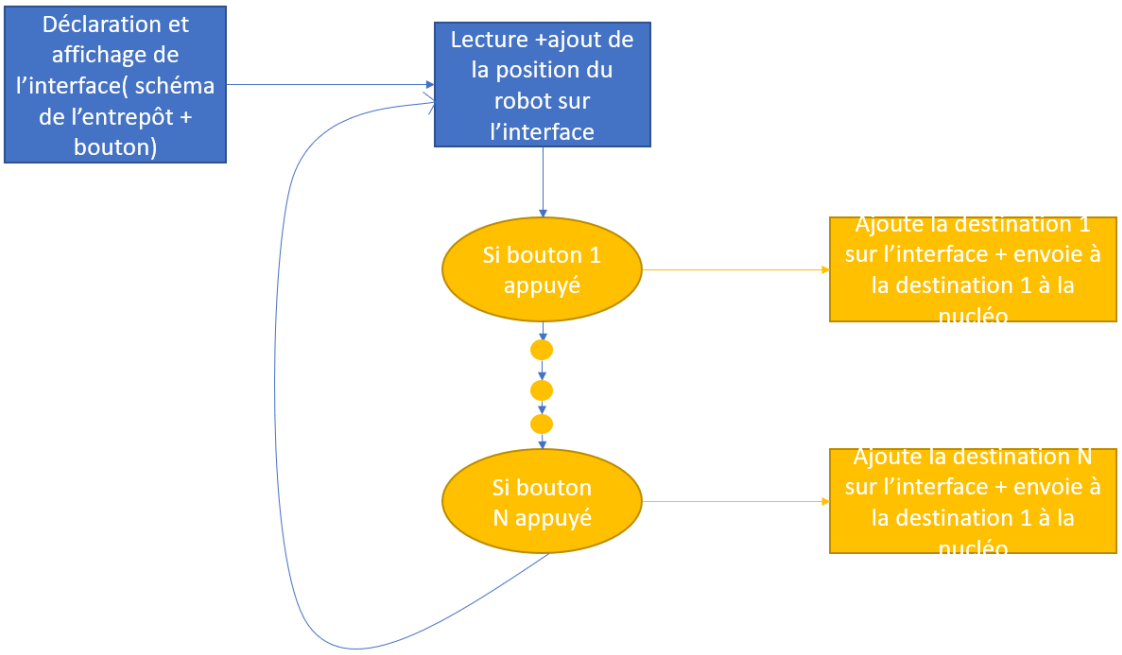
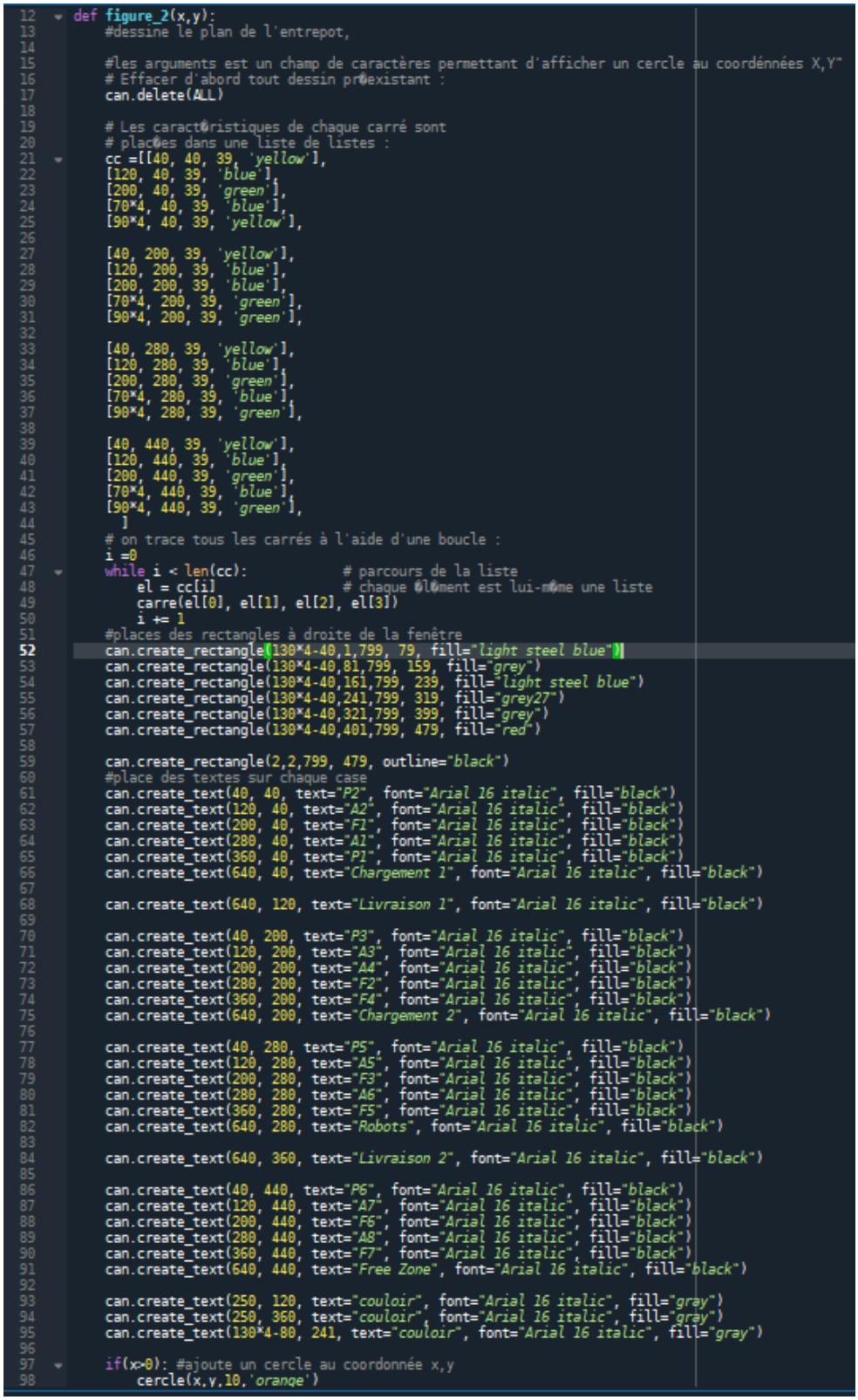



***Figure 2 – Schéma fonctionnel***

Fonctions permettant de crée de cercle et des carrés plus rapidement « can » étant le nom de la fenêtre canvas:

```
1   from tkinter import *
2
3
4   def carre(x, y, r, coul ='black'):
5       "tracé d'un carré de centre (x,y) et de coté 2r"
6       can.create_rectangle(x-r, y-r, x+r, y+r, fill=coul)
7   def cercle(x, y, r, coul ='black'):
8      "tracé d'un cercle de centre (x,y) et de rayon r"
9      can.create_oval(x-r, y-r, x+r, y+r, fill=coul)
```

Cette fonction crée la fenêtre canvas et affiche le plan de l'entrepot ainsi qu'un cercle de centre X,Y



```
def figure_2(x,y):
    #dessine le plan de l'entrepot,

    #les arguments est un champ de caractères permettant d'afficher un cercle au coordénnées X,Y"
    # Effacer d'abord tout dessin pr€existant :
    can.delete(ALL)

    # Les caract€ristiques de chaque carré sont
    # plac€es dans une liste de listes :
    cc =[[40, 40, 39, 'yellow'],
    [120, 40, 39, 'blue'],
    [200, 40, 39, 'green'],
    [70*4, 40, 39, 'blue'],
    [90*4, 40, 39, 'yellow'],

    [40, 200, 39, 'yellow'],
    [120, 200, 39, 'blue'],
    [200, 200, 39, 'blue'],
    [70*4, 200, 39, 'green'],
    [90*4, 200, 39, 'green'],

    [40, 280, 39, 'yellow'],
    [120, 280, 39, 'blue'],
    [200, 280, 39, 'green'],
    [70*4, 280, 39, 'blue'],
    [90*4, 280, 39, 'green'],

    [40, 440, 39, 'yellow'],
    [120, 440, 39, 'blue'],
    [200, 440, 39, 'green'],
    [70*4, 440, 39, 'blue'],
    [90*4, 440, 39, 'green'],
      ]
    # on trace tous les carrés à l'aide d'une boucle :
    i =0
    while i < len(cc):            # parcours de la liste
        el = cc[i]                # chaque €l€ment est lui-m€me une liste
        carre(el[0], el[1], el[2], el[3])
        i += 1
    #places des rectangles à droite de la fenêtre
    can.create_rectangle(130*4-40,1,799, 79, fill="light steel blue")
    can.create_rectangle(130*4-40,81,799, 159, fill="grey")
    can.create_rectangle(130*4-40,161,799, 239, fill="light steel blue")
    can.create_rectangle(130*4-40,241,799, 319, fill="grey27")
    can.create_rectangle(130*4-40,321,799, 399, fill="grey")
    can.create_rectangle(130*4-40,401,799, 479, fill="red")

    can.create_rectangle(2,2,799, 479, outline="black")
    #place des textes sur chaque case
    can.create_text(40, 40, text="P2", font="Arial 16 italic", fill="black")
    can.create_text(120, 40, text="A2", font="Arial 16 italic", fill="black")
    can.create_text(200, 40, text="F1", font="Arial 16 italic", fill="black")
    can.create_text(280, 40, text="A1", font="Arial 16 italic", fill="black")
    can.create_text(360, 40, text="P1", font="Arial 16 italic", fill="black")
    can.create_text(640, 40, text="Chargement 1", font="Arial 16 italic", fill="black")

    can.create_text(640, 120, text="Livraison 1", font="Arial 16 italic", fill="black")

    can.create_text(40, 200, text="P3", font="Arial 16 italic", fill="black")
    can.create_text(120, 200, text="A3", font="Arial 16 italic", fill="black")
    can.create_text(200, 200, text="A4", font="Arial 16 italic", fill="black")
    can.create_text(280, 200, text="F2", font="Arial 16 italic", fill="black")
    can.create_text(360, 200, text="F4", font="Arial 16 italic", fill="black")
    can.create_text(640, 200, text="Chargement 2", font="Arial 16 italic", fill="black")

    can.create_text(40, 280, text="P5", font="Arial 16 italic", fill="black")
    can.create_text(120, 280, text="A5", font="Arial 16 italic", fill="black")
    can.create_text(200, 280, text="F3", font="Arial 16 italic", fill="black")
    can.create_text(280, 280, text="A6", font="Arial 16 italic", fill="black")
    can.create_text(360, 280, text="F5", font="Arial 16 italic", fill="black")
    can.create_text(640, 280, text="Robots", font="Arial 16 italic", fill="black")

    can.create_text(640, 360, text="Livraison 2", font="Arial 16 italic", fill="black")

    can.create_text(40, 440, text="P6", font="Arial 16 italic", fill="black")
    can.create_text(120, 440, text="A7", font="Arial 16 italic", fill="black")
    can.create_text(200, 440, text="F6", font="Arial 16 italic", fill="black")
    can.create_text(280, 440, text="A8", font="Arial 16 italic", fill="black")
    can.create_text(360, 440, text="F7", font="Arial 16 italic", fill="black")
    can.create_text(640, 440, text="Free Zone", font="Arial 16 italic", fill="black")

    can.create_text(250, 120, text="couloir", font="Arial 16 italic", fill="gray")
    can.create_text(250, 360, text="couloir", font="Arial 16 italic", fill="gray")
    can.create_text(130*4-80, 241, text="couloir", font="Arial 16 italic", fill="gray")

    if(x>0): #ajoute un cercle au coordonnée x,y
        cercle(x,y,10,'orange')
```

On utilise ces fonctions qui appelle directement la fonction précedentes, on a besoins de crée ces fontions car elles ont l'avantage de ne pas avoir d'argument d'entrée et donc de pouvoir être assimilé à un bouton.

```
 #ces fonctions permettent d'afficher le plan de avec un point orange à une certaine coordonée
def A2():

    figure_2(120,60)
def P2():
    figure_2(40,60)
def F1():
    figure_2(200,60)
def A1():
    figure_2(280,60)
def P1():
    figure_2(360,60)
def P3():
    figure_2(40,220)
def A3():
    figure_2(120,220)
def A4():
    figure_2(200,220)
def F2():
    figure_2(280,220)
def F4():
    figure_2(360,220)
def P5():
    figure_2(40,300)
def A5():
    figure_2(120,300)
def F3():
    figure_2(200,300)
def A6():
    figure_2(280,300)
def F5():
    figure_2(360,300)
def P6():
    figure_2(40,460)
def A7():
    figure_2(120,460)
def F6():
    figure_2(200,460)
def A8():
    figure_2(280,460)
def F7():
    figure_2(360,460)
def C1():
    figure_2(520,40)

def L1():
    figure_2(520,120)
def C2():
    figure_2(520,200)
def R():
    figure_2(520,280)
def L2():
    figure_2(520,360)
def FZ():
    figure_2(520,440)



#simule la trajectoire d'un robot
```

Cette fonction est ajoutée pour simuler la réception de donnée au cours du temps de la position d'un robot et l'affiche sur la figure. Dans le cas de notre simulation le vecteur x et y sont des nombres prédéfinis pour simuler une trajectoire et montre qu'on peut modifier l'interface dans le temps.

```
#simule la trajectoire d'un robot
def animer(h):

    global x,y # permet davoir x et y dans la fonction



    id1=cercle(x[h],y[h],10,'black')
    can.delete(id1)

    fen.after(3000, animer,h+1)
```

Main programme, on configure la grille de la fenêtre et on ajoute les boutons.

```
##### Programme principal : ############
fen = Tk()
can = Canvas(fen, width =800, height =500, bg ='ivory')
can.grid(row=1, column=1,columnspan=11)
#configure chaque bouton pour effectuer la bonne fonction quand on appuit
b1 = Button(fen, text ='P2', command =P2)
b1.grid(row=2, column=1)
b2 = Button(fen, text ='A2', command =A2)
b2.grid(row=2, column=2)
b3 = Button(fen, text ='F1', command =F1)
b3.grid(row=2, column=3)
b4 = Button(fen, text ='A1', command =A1)
b4.grid(row=2, column=4)
b5 = Button(fen, text ='P1', command =P1)
b5.grid(row=2, column=5)

b6 = Button(fen, text ='P3', command =P3)
b6.grid(row=4, column=1)
b7= Button(fen, text ='A3', command =A3)
b7.grid(row=4, column=2)
b8 = Button(fen, text ='A4', command =A4)
b8.grid(row=4, column=3)
b9 = Button(fen, text ='F2', command =F2)
b9.grid(row=4, column=4)
b10 = Button(fen, text ='F4', command =F4)
b10.grid(row=4, column=5)

b11 = Button(fen, text ='P5', command =P5)
b11.grid(row=5, column=1)
b12 = Button(fen, text ='A5', command =A5)
b12.grid(row=5, column=2)
b13 = Button(fen, text ='F3', command =F3)
b13.grid(row=5, column=3)
b14 = Button(fen, text ='A6', command =A6)
b14.grid(row=5, column=4)
b15 = Button(fen, text ='F5', command =F5)
b15.grid(row=5, column=5)

b16 = Button(fen, text ='P6', command =P6)
b16.grid(row=7, column=1)
b17= Button(fen, text ='A7', command =A7)
b17.grid(row=7, column=2)
b18 = Button(fen, text ='F6', command =F6)
b18.grid(row=7, column=3)
b19 = Button(fen, text ='A8', command =A8)
b19.grid(row=7, column=4)
b20 = Button(fen, text ='F7', command =F7)
b20.grid(row=7, column=5)

b21 = Button(fen, text ='Chargement 1', command =C1)
b21.grid(row=2, column=10)
b22 = Button(fen, text ='livraison 1', command =L1)
b22.grid(row=3, column=10)
b23 = Button(fen, text ='Chargement 2', command =C2)
b23.grid(row=4, column=10)
b24 = Button(fen, text ='Robots', command =R)
b24.grid(row=5, column=10)
b25 = Button(fen, text ='livraison 2', command =L2)
b25.grid(row=6, column=10)
b26 = Button(fen, text ='Free Zone', command =FZ)
b26.grid(row=7, column=10)
figure_2(-1,-1)
#trajectoire simulé du robot
x=[520,440,440,440,440,360,360]
y=[360,360,320,250,120,120,60]
animer(0)
fen.mainloop()
```

## *IV – Simulation du moteur sous Matlab :*

### A) Description

On a modélisé ce système sur Matlab avec l'application Simulink et l'add-on Simscape. Ce système modélisé se construit sous forme d'un schéma bloc :

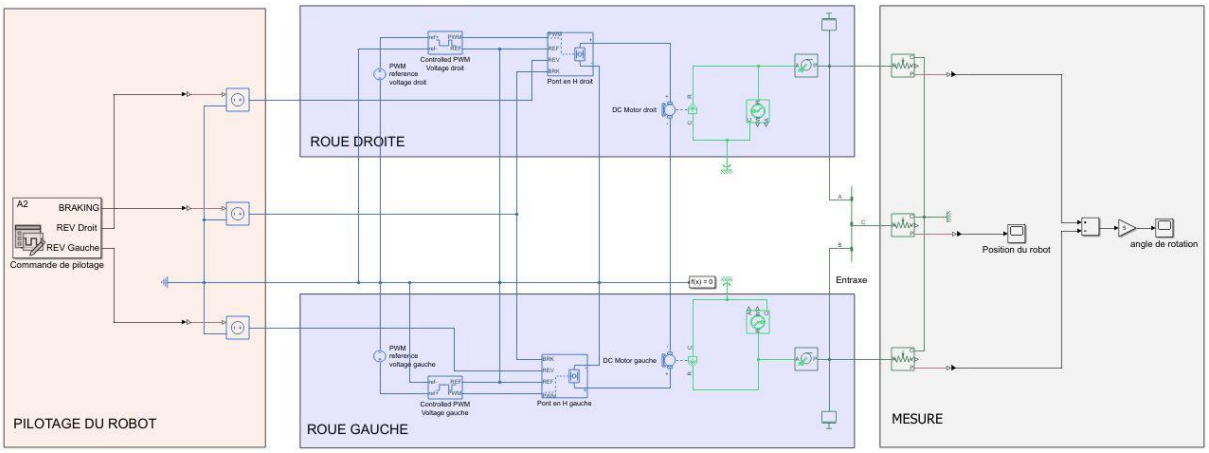



**Figure 1 ; Schéma bloc général du robot piloté**

Le robot est composé trois parties : une partie pilotage, une partie mécanique et une dernière partie permettant la mesure des variables intéressantes comme la distance parcourue par le robot ou son angle d'orientation.

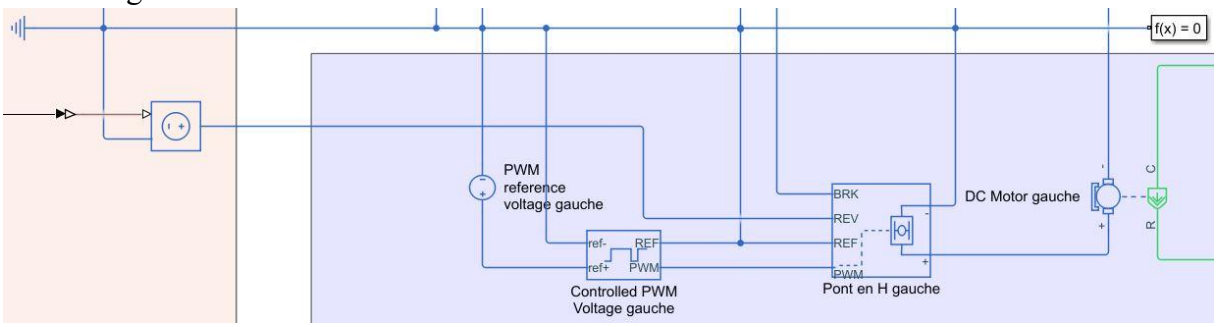



**Figure 2 ; zoom sur la partie moteur à courant continu**

La partie mécanique est divisée elle-même en plusieurs parties :

- 2 moteurs à courant continu actionnés en PWM
- 2 roues auxquelles nous avons ajouté une masse de 1 Kg pesant sur l'axe
- 1 entraxe de 20 cm permettant de relier les deux roues du robot

La figure 2 montre en détail les éléments composant le moteur à courant continu contrôlé en PWM. Le DC moteur est relier à un pont en H qui nous permet de stopper le moteur ainsi qu'inverser son sens de rotation. Avant ce bloc, on a une création d'un signal PWM de fréquence 4000 Hz entre 0 et 5 V, dont le rapport cyclique est contrôlé par un générateur. Le générateur permet donc de contrôler la vitesse de la roue.

### B) Pilotage du Robot

1) Valeur de la vitesse linéaire du robot

Pour respecter le cahier des charges du robot on ne doit pas dépasser une vitesse linéaire de 0.3 $m.s^{-1}$. On teste donc différentes valeurs de rapport cyclique de la PWM pour obtenir cette vitesse. Sur la simulation on ne rentre pas la valeur du rapport cyclique mais directement la tension moyenne de la PWM.
On mesure la vitesse linéaire du robot grâce à un capteur nommé « Ideal Translational Motion Sensor » et un oscillateur pour afficher la courbe (Figure 1).

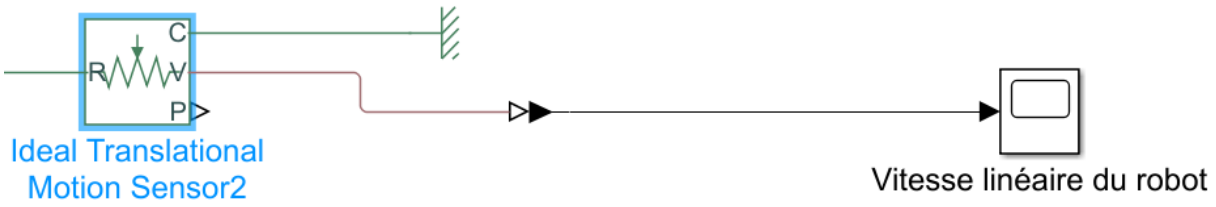



**Figure 1 : Schéma bloc de mesure de la vitesse linéaire**

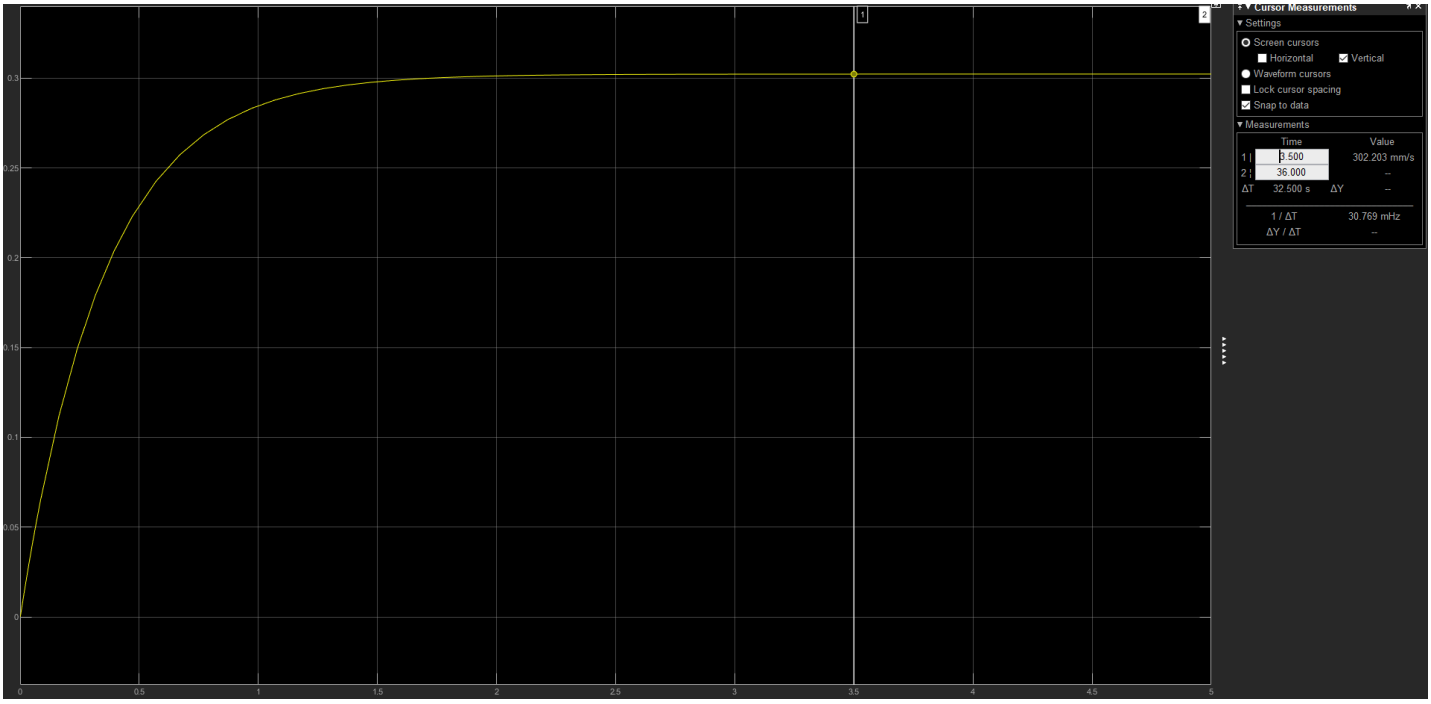



**Figure 2 : Courbe de vitesse en fonction du temps**

Selon notre modélisation du moteur il faudra une tension moyenne de PWM égale à 0,78 V (sur 5V) pour obtenir une vitesse linéaire de 0,3 $m.s^{-1}$.

2) Méthode de pilotage du robot

Pour piloter le robot nous garderons toujours la même valeur de la PWM pour les deux moteurs des deux roues ainsi de base les deux moteurs des roues seront constamment en fonctionnement et comme ils tournent à la même vitesse puisqu'ils ont la même valeur de tension de PWM le robot avance en ligne droite. Nous piloterons l'avancé et la rotation du robot grâce aux deux hacheurs des roues droite et gauche du robot. Ces hacheurs ont des entrées BRAKE et REVERSE qui lorsqu'on leurs impose une tension supérieure à un certain seuil (5V dans notre

cas) font que le moteur s'arrête ou fonctionne en sens inverse. L'objectif sera donc d'envoyer des signaux logiques (sous forme de tension 0V ou 5V) aux entrées BRAKE et REVERSE à des temps bien précis afin que le robot avance de la distance souhaitée et tourne de l'angle voulu suivant la trajectoire à effectuer.

3) Pilotage du robot en ligne droite

Pour avancer en ligne droite d'une certaine distance nous utiliserons l'entrée BRAKE. On envoie tout d'abord une tension nulle sur l'entrée au niveau des deux moteurs, ainsi les deux moteurs fonctionnent et le robot avance en ligne droite. Lorsque nous devons arrêter le robot on impose une tension de 5V sur l'entrée BRAKE pour les deux moteurs, le robot s'arrête.
Cependant à cause de l'inertie du robot nous sommes obligés de déterminer le temps du régime transitoire pour le démarrage et pour l'arrêt.
D'après nos simulations nous avons trouvé que le régime stationnaire était obtenu à partir de 1,62 s pour l'arrêt et le démarrage.
Durant le régime transitoire au démarrage le robot parcourt 0,384m et 0,100m pour l'arrêt.
Ainsi nous devrons veiller à stopper les moteurs des deux roues 1,62s avant l'arrêt du robot, soit 0,100m avant la position finale.
Sinon en régime permanent le robot va à une vitesse de 0,3 $m.s^{-1}$.
Ainsi pour faire avancer le robot d'un mètre il faudra couper les moteurs à partir de 3,327s.
Pour faire deux mètres il faudra les couper au bout de 6,639s.
Et finalement pour faire k mètres avec k > 2 il faudra couper les moteurs à partir de 6,639 + k*3,309 s.

4) Pilotage du robot en rotation

Pour effectuer une rotation de 90° à gauche (resp. droite) nous utiliserons l'entrée REVERSE du hacheur gauche (resp. droite). En effet pour modéliser la rotation nous inversons le sens de rotation d'une seule roue en gardant toujours les mêmes vitesses de rotations. Ainsi le robot effectue une rotation autour de son centre de masse. En faisant ainsi le logiciel Matlab ne perçoit pas cette rotation comme un changement de position du robot qui viendrait fausser nos valeurs de position.
Tout comme pour la translation nous avons de l'inertie. Ainsi pour effectuer un angle de 90° à droite ou à gauche il faudra arrêter les moteurs avant la fin de la rotation.
Pour effectuer un angle de 90° nous devrons arrêter les moteurs (avec la fonction BRAKE) au bout de 0,52s.

5) Mesure de l'angle et de la position

Finalement nous devrons extraire les position et angle du robot, pour cela nous utilisons les mêmes capteurs que précédemment (Figure 1) mais sur la sortie position au lieu de vitesse.
Pour mesurer la position angulaire nous effectuons une différence entre les distances algébriques parcourues par chacune des roues (un capteur par roue). Cette différence est ensuite divisée par la longueur de l'entraxe entre les deux roues afin d'obtenir la position angulaire.

## C) Résultats

Pour illustrer notre méthode de pilotage nous avons choisi une trajectoire partant de la case : Robot et allant jusqu'à la case A2 (Figure 3). Pour des soucis de simplicité nous avons considéré qu'une case mesurait 1m par 1m.

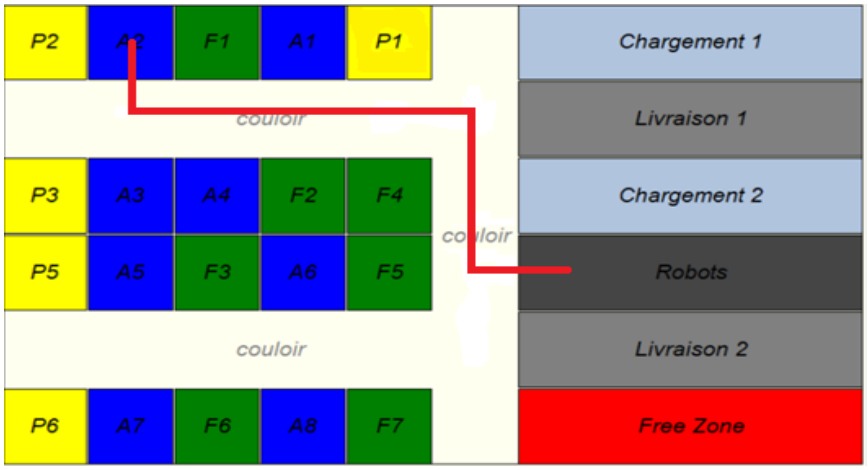



**Figure : Trajectoire utilisée pour notre simulation**

La séquence à réaliser pour suivre cette trajectoire est donc : Avancer d'une case – Tourner à droite – Avancer de deux cases – Tourner à gauche – Avancer de quatre cases – Tourner à droite – Avancer d'une case.
On commence par la fonction BRAKE :

| TEMPS (s) | TENSION (V) | Fonction |
|---|---|---|
| 0 | 0 | Début : Avancer d'une case |
| 3.327 | 0 | |
| 3.327 | 5 | |
| 5 | 5 | Fin : Avancer d'une case |
| 5 | 0 | Début : Tourner à droite |
| 5.52 | 0 | |
| 5.52 | 5 | |
| 8 | 5 | Fin : Tourner à droite |
| 8 | 0 | Début : Avancer de deux cases |
| 14.639 | 0 | |
| 14.639 | 5 | |
| 18 | 5 | Fin : Avancer de deux cases |
| 18 | 0 | Début : Tourner à gauche |
| 18.52 | 0 | |
| 18.52 | 5 | |
| 21 | 5 | Fin : Tourner à gauche |
| 21 | 0 | Début : Avancer de quatre cases |
| 34.257 | 0 | |
| 34.257 | 5 | |
| 38 | 5 | Fin : Avancer de quatre cases |
| 38 | 0 | Début : Tourner à droite |
| 38.52 | 0 | |
| 38.52 | 5 | |

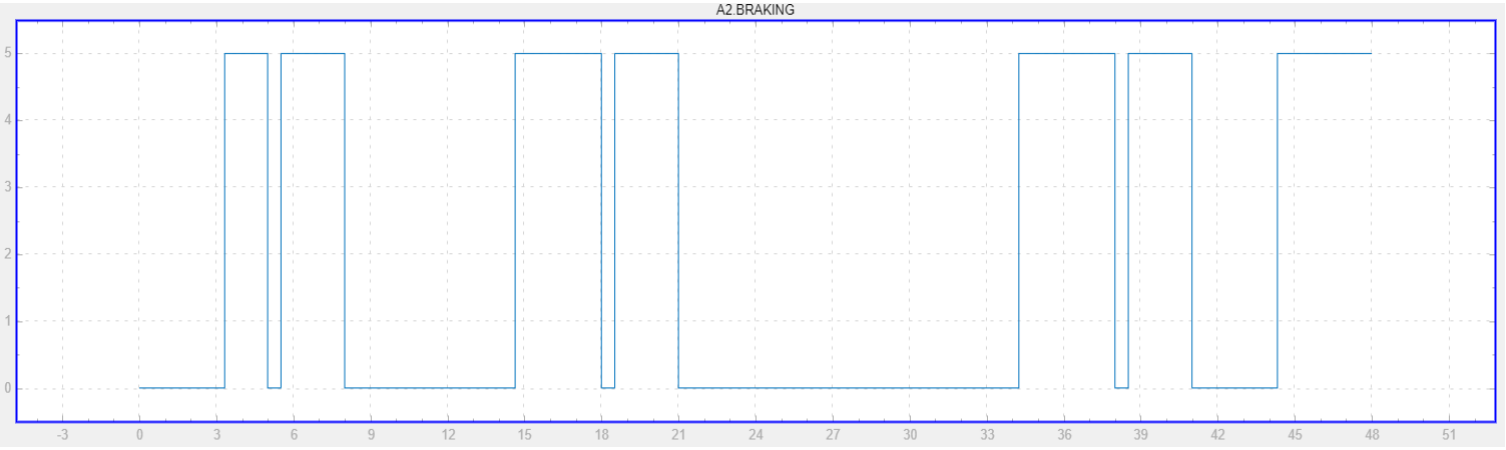



Figure 4 : Tension de commande envoyée sur les entrées BRAKE des deux hacheurs

| | | |
|---|---|---|
| 41 | 5 | Fin : Tourner à droite |
| 41 | 0 | Début : Avancer d'une case |
| 44.327 | 0 | |
| 44.327 | 5 | |
| 48 | 5 | Fin : Avancer d'une case |

Puis la fonction REVERSE pour la roue droite :

| TEMPS (s) | TENSION (V) | Fonction |
|---|---|---|
| 0 | 0 | |
| 5 | 0 | |
| 5 | 5 | Début : Tourner à droite |
| 5.52 | 5 | |
| 5.52 | 0 | Fin : Tourner à droite |
| 38 | 0 | |
| 38 | 5 | Début : Tourner à droite |
| 38.52 | 5 | |
| 38.52 | 0 | Fin : Tourner à droite |
| 48 | 0 | |

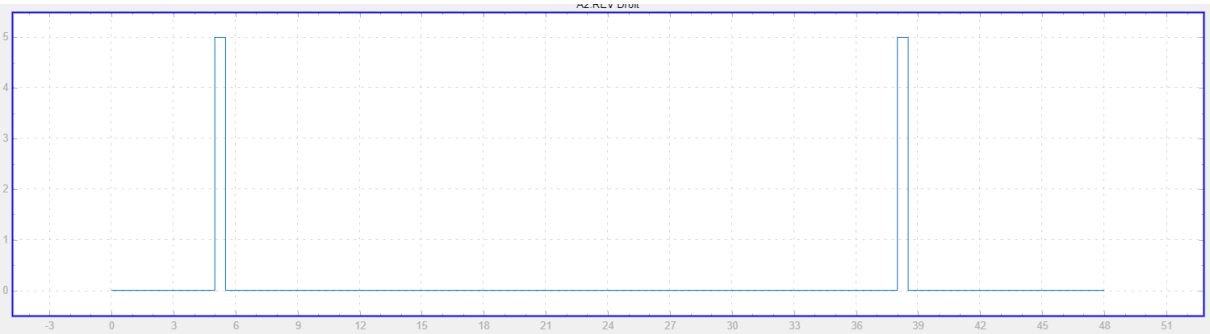


Figure 5 : Tension de commande pour la fonction REVERSE envoyée au hacheur de la roue droite

En finalement la fonction REVERSE pour la roue gauche :

| TEMPS (s) | TENSION (V) | Fonction |
|---|---|---|
| 0 | 0 | |
| 18 | 0 | Début : Tourner à gauche |
| 18 | 5 | |
| 18.52 | 5 | |
| 18.52 | 0 | Fin : Tourner à gauche |
| 48 | 0 | |

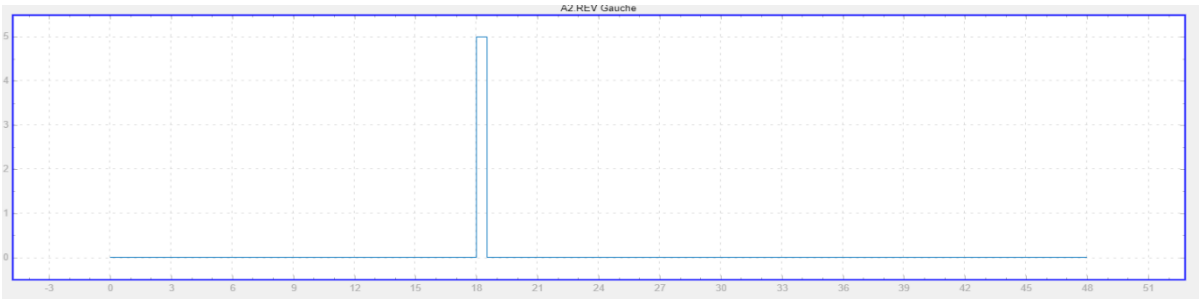



**Figure 3 : Tension de commande pour la fonction REVERSE envoyée au hacheur de la roue droite**

Ce que l'on obtient finalement ce sont les courbes en position et angle du robot au cours du temps ce qui nous permet de vérifier avec le plan que le robot suit bien la bonne trajectoire.

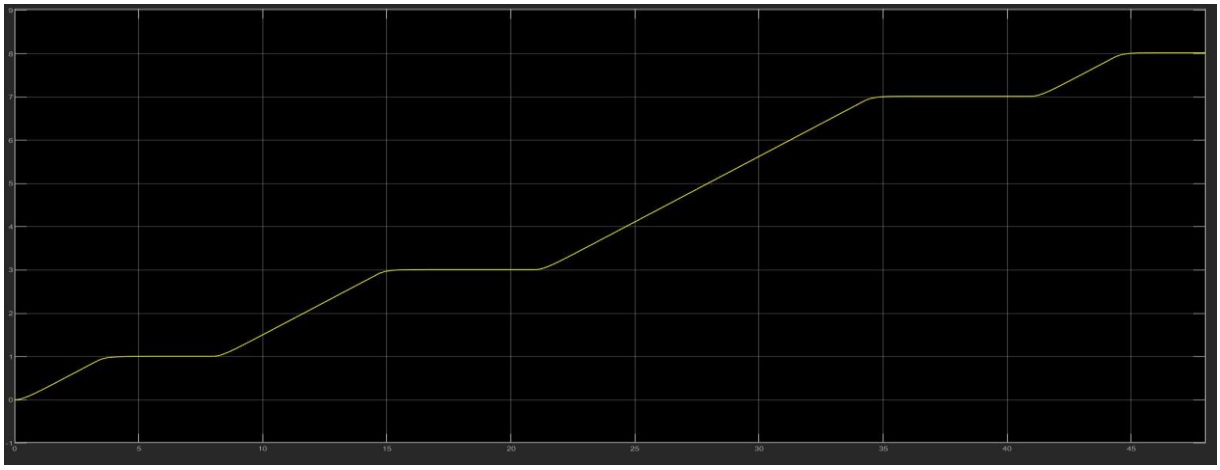


**Figure 7 : Courbe de la distance parcourus par le robot en fonction du temps**

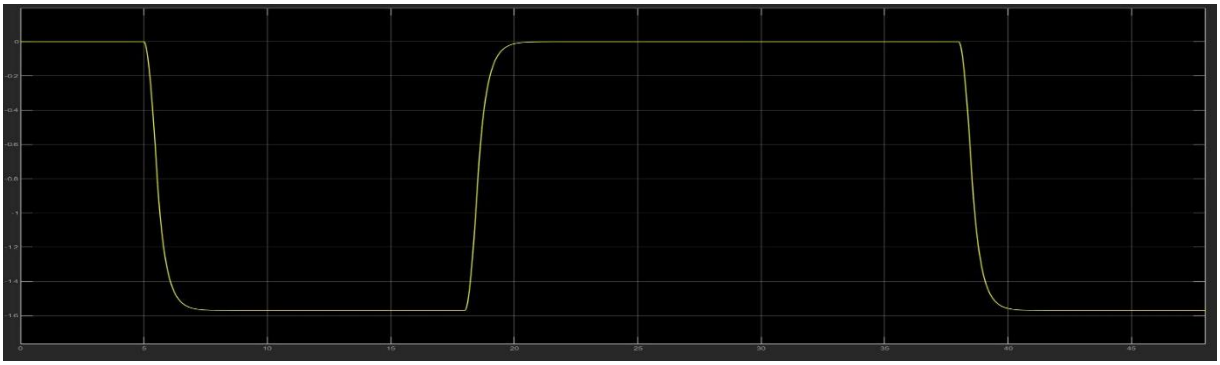


**Figure 8 : Courbe de l'angle du robot par rapport à la référence de départ en fonction du temps**

## D) Limites de la méthode :

Cette modélisation a pour principale limite la connaissance précise de l'ensemble des caractéristiques du système tel que le poids du robot, la taille de l'entraxe, le couple du moteur, le diamètre des roues … et n'est pas robuste du tout puisqu'au moindre changement d'une seule de ces caractéristiques il faudrait modifier l'ensemble des courbes de commande du hacheur.

De plus le manque d'asservissement en position et en vitesse ferait qu'une simple variation des caractéristiques dans la réalité le système divergerait en position et angle, problème que nous ne rencontrons pas dans la simulation puisque tout est considéré « parfait ».
Mais pour une simulation avec des caractéristiques bien fixées dès le départ, cette méthode semble la plus rapide et simple à mettre en place avec une connaissance superficielle de Simulink.