REALISATION D'UN BRAS DE ROBOT D'AIDE AUX PERSONNES À MOBILITÉ RÉDUITE

Auteurs :
AIVAZIAN Inès
HUSSAIN Warith
GAUTIER Paul

Encadrants :
FERSZTEROWSKI Antoine
CALDAS Alex

30 mai 2021

Table des matières

Liste des figures

Remerciements

Tout d'abord, il nous semble évident de remercier Monsieur CALDAS et Monsieur FERSZTEROWSKI pour le temps et l'énergie qu'ils ont consacrés à nous encadrer et à nous guider tout au long de ce projet. Nous les remercions également pour avoir mis à notre disposition le matériel de l'école, afin de réaliser au mieux notre projet.
Malgré la situation actuelle, Monsieur FERSZTEROWSKI a su se rendre disponible au maximum via teams pour nous aider.

Nous tenons ensuite à remercier Madame THERY et les élèves de la Maker Team de nous avoir donné l'accès au FabLab, notamment pour l'impression 3D de nos pièces.

Nous remercions aussi quelques élèves de l'ESME Sudria qui ont pu nous apporter une aide ponctuelle sur la partie théorique du module mécanique.

Pour finir, nous remercions tous nos professeurs de nous avoir enseigné le savoir essentiel qui nous a permis de mener à bien ce travail.

1. Présentation du projet

1.1 Introduction

Que ce soit à la suite d'une expérience fortuite ou de longues études sur un sujet, l'Homme essaie toujours d'améliorer sa qualité de vie. En commençant par la maitrise du feu, il est aujourd'hui dans l'ère du numérique. Pour effectuer une tâche plus rapidement ou avec plus de précision par exemple, bon nombre de processus sont automatisés, à l'instar des chaines de production automobiles, grâce à la robotique.

Un robot permet d'effectuer, grâce à un système de commande automatique à base de micro-processeur, une tâche précise pour laquelle il a été conçu dans le domaine industriel, scientifique, militaire ou domestique. Dans le cadre de notre projet, nous souhaitons nous focaliser sur l'aspect médical et essayer de venir en aide aux personnes à mobilité réduites. En effet, en France, plus de 8 millions de personnes déclarent avoir une ou plusieurs déficiences motrices, ce qui représente environ 13% de la population.

Nous avons donc décidé de consacrer notre réflexion à la création d'un bras robotisé pour venir en aide aux personnes qui subissent une motricité difficile, et parfois impossible.

Le premier bras articulé (Unimate) est créé en 1954 par l'américain George Devo dans le but d'être utilisé dans l'industrie nucléaire.
Depuis, les bras robotisés se sont propagés à tous les domaines et permettent de faire de nombreuses tâches : déplacer de lourds objets, aider les chirurgiens lors d'opérations complexes, récupérer des échantillons sur des astéroïdes…

Notre objectif est donc de répondre à la problématique suivante : **Comment pourrions-nous aider les personnes à mobilité réduite, grâce à nos connaissances et aux acquis de la technologie ?**

Dans ce cahier des charges, nous aborderons plusieurs parties. Dans un premier temps, nous présenterons le projet dans sa globalité. Ensuite, nous expliquerons en détail les trois grands modules qu'il comporte : l'informatique, l'électronique et la mécanique. Pour finir, nous étudierons l'aspect management d'un tel processus.

1.2 Liste des exigences

Le but de ce projet est de réaliser un bras robotisé pouvant bouger sur trois axes, muni d'une pince et pouvant attraper des objets du quotidien plus ou moins lourds. Pour ce faire, plusieurs exigences nous ont été données :

Déplacement

- Le système doit avoir 3 degrés de liberté
- Le système doit être contrôlable grâce à une manette
- Le système doit déplacer un objet d'un point A à un point B
- Le système doit effectuer des mouvements avec fluidité

Équipements

- Le système doit être muni d'une pince
- Cette pince doit permettre la saisie d'un objet non fragile

Technique

- L'ensemble des pièces mécaniques seront à imprimer à l'imprimante 3D
- Les pièces seront modélisées sur Solidworks
- Le système sera contrôlé par raspberry

Livrable

- Le code et la documentation seront livrés sous forme de répertoire GitHub
- L'ensemble du livrable permettra à toute personne extérieure au projet de le reproduire

1.3 Schéma global

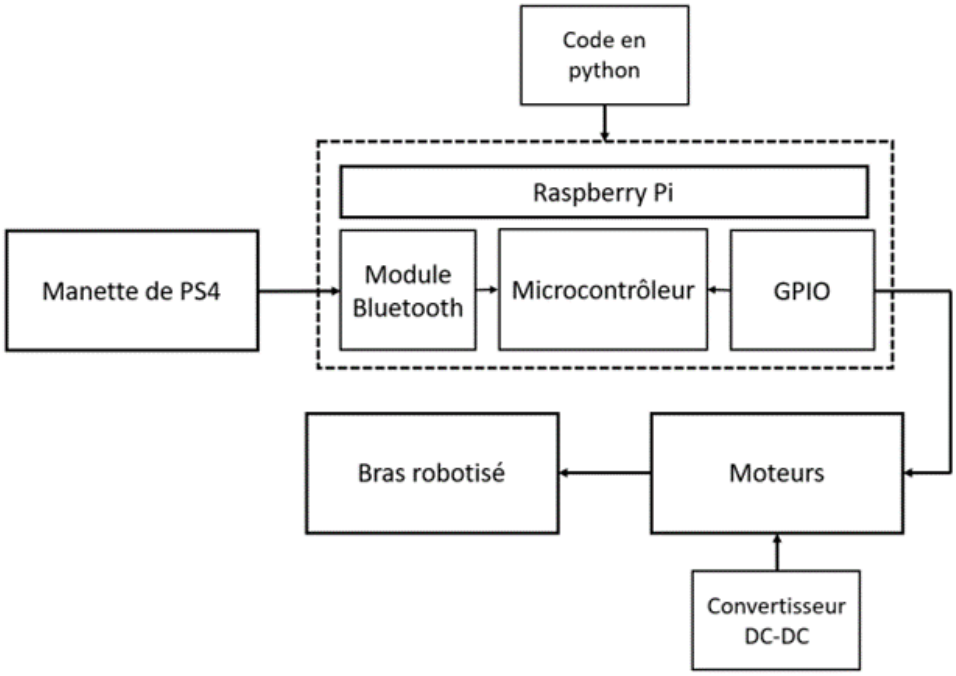


Figure 1 : Schéma global du projet

Tout d'abord, nous avons utilisé une manette de PS4 qui envoie les données utilisées par Bluetooth à la carte Raspberry Pi. Celle-ci s'occupe de traiter ces données grâce à un code implémenté en python. Ensuite, les moteurs deviennent pilotables en activant les joysticks et bouton de la manette. Enfin, nous avons utilisé un convertisseur DC-DC qui permet d'alimenter les servomoteurs.

Avant de débuter le projet, nous nous sommes préparés à rencontrer des difficultés, que ça soit au niveau du code comme au niveau de la modélisation et de l'impression 3D. Cependant, nos attentes vis-à-vis de ce projet reposent sur des conditions exigées et nous nous sommes efforcés de mener au mieux possible ce travail d'équipe.

2. Présentation du projet

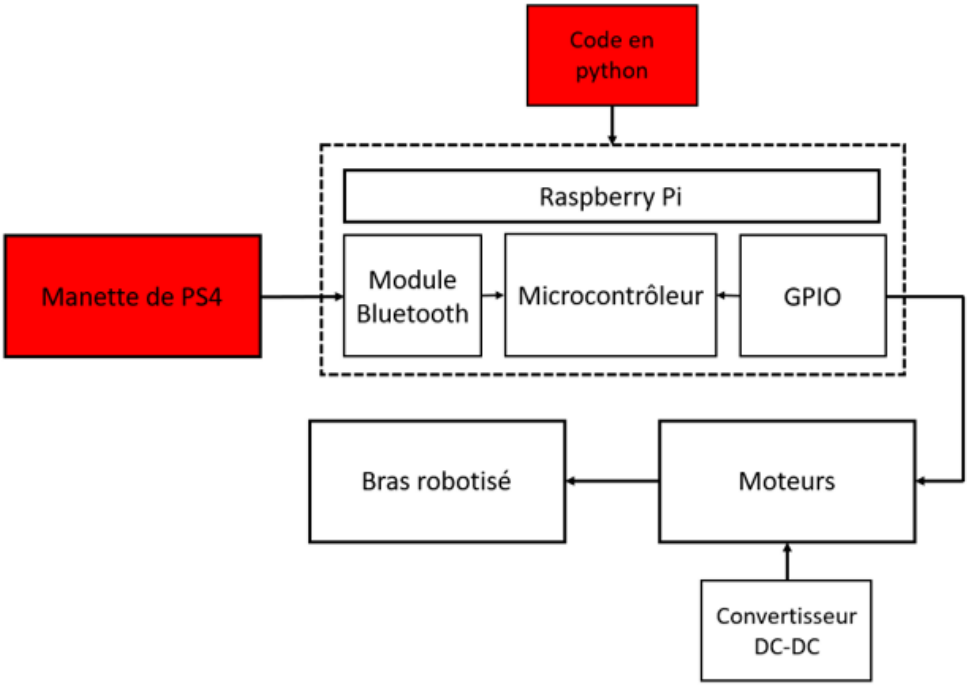


Figure 2 : Schéma global du projet (partie informatique)

Le code final du projet est séparé en 4 blocs distincts : la configuration du moteur, le mappage de la manette de PS4, la configuration et le mappage du keypad, et enfin le code main faisant appel aux fonction nécessaires.

C'est donc dans cet ordre bien défini que la partie informatique sera détaillée avec des captures d'écran pour chaque sous-partie.

2.1 Configuration moteur

Les servomoteurs sont contrôlés en position à l'aide d'une PWM qui est expliquée plus en détails dans la partie Électronique. La carte Raspberry Pi ayant 2 pins PWM hardware, nous utilisons des GPIOs basiques pour générer une SoftPWM.
Nos moteurs sont des servos digitaux et il est donc possible d'augmenter la fréquence afin d'améliorer la précision des positions. Dans ce cas, nous avons utilisé une fréquence de 300 Hz.

```
1   import RPi.GPIO as GPIO
2   import time
3
4   class MotorControl:
5       def __init__(self,gpio_motor,initpos=0,pos_min=15,pos_max=75,frequency=300):
6           GPIO.setmode(GPIO.BCM)
7
8           #Motor1
9           #variable set
10          self.pos_min=pos_min
11          self.pos_max=pos_max
12
13          #PWM Intialisation
14          servo_pin = gpio_motor
15          GPIO.setup(servo_pin, GPIO.OUT)
16          self.motor = GPIO.PWM(servo_pin, frequency)
17          self.motor.start(initpos)
18          self.position=initpos
19
20          #Put the PWM to 0 so the motor doesn't vibrate
21          if initpos != 0:
22                  time.sleep(1)
23                  self.motor.ChangeDutyCycle(0)
24
```

Figure 3 : Code configuration moteur (1)

La commande *Import RPI.GPIO* permet d'importer une librairie qui est nécessaire afin d'utiliser les GPIOs de la Raspberry Pi.
Pour ce projet, nous avons utilisé 4 moteurs avec une configuration presque similaire, et donc coder des classes et des méthodes afin de les configurer et les contrôler. Les objets de chaque moteur seront créés dans le module principal à l'aide des paramètres situés dans le constructeur.
Les moteurs ont chacun un pin assigné (sinon ils vont tourner en même temps), on met donc en paramètre le pin du moteur, ici *gpio_motor*.

Puis, nous avons initialisé 3 autres paramètres qui ont une valeur par défaut.

- La position initiale, qui est à 0 par défaut qui signifie que le moteur ne tournera pas et ne prendras pas de position en particulier.
- La position minimale, qui est à 15 par défaut qui correspond à 0°.
- La position maximale, qui est à 75 qui correspond à 180°

Lorsque l'on donne une position initiale au moteur (*initpos* différent de 0), on laisse le temps au moteur de prendre la position, et on remet la PWM à 0, à l'aide de la fonction *ChangeDutyCycle*, pour stopper les tremblements du moteur.

```
26      def move_motor(self,value):
27          self.motor.ChangeDutyCycle(value)
28
29      def move_max1(self):
30          self.motor.ChangeDutyCycle(self.pos_max)
31
32      def move_min1(self):
33          self.motor.ChangeDutyCycle(self.pos_min)
34
35      def move_max(self):
36          if self.position<self.pos_max :
37              self.position+=1
38          self.motor.ChangeDutyCycle(self.position)
39          time.sleep(0.01)
40          print(self.position)
41
42      def move_min(self):
43          if self.position>self.pos_min :
44              self.position-=1
45          self.motor.ChangeDutyCycle(self.position)
46          time.sleep(0.01)
47          print(self.position)
48
49      def stop_now(self):
50          self.motor.ChangeDutyCycle(0)
```

Figure 4 : Code configuration moteur (2)

Nous avons créé des méthodes et des classes afin de contrôler le moteur ; pour cela, il y a 3 façons de procéder :

- *Move_motor* qui prend en paramètre une valeur (entre la position min et la position max) afin d'atteindre une position voulue. Cette méthode est pratique lorsque nous voulons atteindre une position précise.
- *Move_min1* et *Move_max1* qui permettent d'atteindre la position minimale et la position maximale. Mais associer à la fonction *stop_now*, qui elle remet la PWM à 0, nous pouvons arrêter le moteur avant qu'elle n'atteigne la position.
  Cette méthode fonctionne seulement pour les servos Moteur MG996R en high torque. La raison de se fonctionnement et la différence des moteurs testé sera expliquée dans la partie électronique.
- *Move_min* et *Move_max* sont des fonctions qui font incrémenter et décrémenter la variable de position et la renvoie au moteur pour changer la position et la maintenir. Sachant qu'en maintenant une PWM à une valeur particulière le moteur peux avoir des interférences et tremblé. On peut donc mettre la PWM à 0 avec *stop_now,* pour stopper le moteur.

En stoppant le moteur, le moteur devient hors tension, et perd donc de sa puissance. Avec le poids du bras, le moteur tourne et fais rabaisser les bras. Il ne faut donc pas mettre la PWM à 0 afin de maintenir le bras en position voulu.

2.2 Mappage de la manette de PS4

La manette de PS4 se connecte à la Raspberry Pi via le module Bluetooth interne à la carte, et nous récupérons les données de la manette grâce à la librairie pyPS4Controller.
Dans la librairie, la class Controller est celle qui initialise la connexion. Quand une manette est connectée, elle à un fichier dans ce chemin '/dev/input/js0' il faut donc appeler la class Controller avec en paramètre « interface = » le chemin.
Avec la méthode « listen », on attend que la manette se connecte. Elle vérifie si le fichier existe dans le chemin donné pendant un temps timeout qui vaut 30s par défaut.
Puis on écoute quelle touche est appuyée, qui va faire lancer les méthodes de la Classe Action dépendant des touches appuyées.
Jusque-là nous avons expliqué comment fonctionne la librairie ; dans notre code, on importe la librairie et on surcharge la class Action afin de pouvoir donner de nouvelles actions à nos touches ce qui va nous permettre de contrôler les moteurs.

```
1  import RPi.GPIO as GPIO
2  import time
3
4  from pyPS4Controller.controller import Controller
5  from Motor_Config import MotorControl
6
7  import os
8
9
10 class MyController(Controller):
11
12     def __init__(self,motors_dico, **kwargs):
13         Controller.__init__(self, **kwargs)
14         self.MC=motors_dico
15
```

Figure 5 : Code mappage de la manette de PS4 (1)

Nous importons donc la librairie mais aussi notre class de Configuration Moteur, afin d'utiliser les méthodes pour les contrôler.
Dans notre Module principal qui fait exécuter tout notre programme, nous avons créé un dictionnaire qui va regrouper les 4 moteurs. Nous mettons donc un dictionnaire en paramètre "motors_dico" et le mettons dans une variable de classe Self.MC.

```
17      def on_L1_press(self):
18          self.MC["motor_pince"].move_min1()
19
20      def on_L1_release(self):
21          self.MC["motor_pince"].stop_now()
22
23      def on_R1_press(self):
24          self.MC["motor_pince"].move_max1()
25
26      def on_R1_release(self):
27          self.MC["motor_pince"].stop_now()
28
```

Figure 6 : Code mappage de la manette de PS4 (2)

Nous utilisons les boutons R1 et L1 de la manette de PS4 pour serrer ou desserrer la pince. Comme vous pouvez le voir, on appelle le dictionnaire stocker dans Self.MC, on annonce lequel des moteurs on souhaite contrôler, dans ce cas "*motor_pince*", puis pour la pince. Puisque nous avons mis les servomoteurs en rotation continue, ils sont contrôlables uniquement par le sens et la vitesse. Nous utilisons donc la méthode *move_max1* et *move_min1* pour choisir le sens de rotation et nous stoppons le moteur lorsque l'on relâche le bouton.

```
31      def on_circle_press(self):
32          print("on_circle_press")
33          self.MC["motor_base"].move_min1()
34
35      def on_circle_release(self):
36          print("on_circle_release")
37          self.MC["motor_base"].stop_now()
38
39      def on_square_press(self):
40          print("on_square_press")
41          self.MC["motor_base"].move_max1()
42
43      def on_square_release(self):
44          print("on_square_release")
45          self.MC["motor_base"].stop_now()
```

Figure 7 : Code mappage de la manette de PS4 (3)

Vu que nous avons choisi de mettre les moteurs à rotation continue sur la base et la pince, pour le moteur de la base nous utilisons le même principe que pour la pince. Ici, nous avons utilisé les boutons rond et carré pour faire tourner le bras par rapport à la base.

Le code pour les articulations 1 et 2 sont toujours en phase de test.

Le souci est que lorsqu'un moteur n'est plus fourni en PWM (PWM=0) il n'est plus sous tension, or un moteur hors tension ne possède plus ses capacités de couples. Et à ce moment-là, il ne maintient plus le bras dans sa position. C'est pourquoi nous utilisons les méthodes move_max et move_min pour ces moteurs afin de les contrôler en position et de maintenir la position avec la PWM constamment allumée.

```
50      def on_L3_up(self, value):
51          self.MC["motor_art_1"].move_max()
52
53      def on_L3_down(self, value):
54          self.MC["motor_art_1"].move_min()
55      """
56      def on_L3_y_at_rest(self):
57          print("on_L3_y_at_rest")
58          self.MC["motor_art_1"].stop_now()
59
60      def on_L3_x_at_rest(self):
61          print("on_L3_x_at_rest")
62          self.MC["motor_art_1"].stop_now()
63      """
```

Figure 8 : Code mappage de la manette de PS4 (4)

Comme vous pouvez le voir ici, nous avons commenté la partie où nous mettons la PWM à 0 car sinon le bras retombe.
Nous utilisons l'axe Y du joystick gauche de la manette pour contrôler la première articulation. À savoir que les joysticks renvoient des valeurs tant qu'ils changent de position. Les méthodes de déplacement du moteur ne font qu'incrémenter et décrémenter seulement lorsque nous changeons la position. La position étant très précise, tant qu'on ne met pas le joystick tout en haut ou tout en bas, la position ne reste pas stable donc la méthode se relance constamment et fait donc incrémenter et décrémenter en continue tant qu'on ne relâche pas le joystick.

```
66  #R3 n'est pas mappé correctement dans la librairie,
67  #on reçois donc les valeurs de R3 sur les methodes de R2
68      def on_R2_press(self, value):
69          if value>0:
70              self.MC["motor_art_2"].move_min()
71          elif value<0 :
72              self.MC["motor_art_2"].move_max()
73        #  elif value==0 :
74        #      self.MC["motor_art_2"].stop_now()
75
76          print("on_R2_press: {}".format(value))
```

Figure 9 : Code mappage de la manette de PS4 (5)

Les gâchettes R2 et L2 renvoient aussi des valeurs en fonction de la pression sur la touche. Mais le mappage n'ayant pas été bien fait dans la librairie, on_R2_press correspond en fait à R3_up et R3_down. Lorsque l'on baisse le joystick droit la méthode on_R2_press retourne une valeur négative et lorsque l'on monte le joystick, une valeur positive est retournée. Et lorsqu'on relâche le joystick, value vaut 0. On vérifie donc la valeur retournée et on fonction de son signe on incrémente ou décrémente sa position. Et lorsque value vaut 0, on ne fait rien pour maintenir la position donnée.

```
78      def on_options_press(self):
79          os.system("bash /home/pi/disconnect.sh")
```

Figure 10 : Code mappage de la manette de PS4 (6)

Lorsque nous appuyons sur le bouton option, on lance le script disconnect.sh qui fais déconnecter la manette de la Raspberry Pi. Ceci permet de redonner la main au Keypad.

2.3 La configuration et mappage du Keypad

Pour le Keypad, il existe des librairies sur internet mais voulant un parfait contrôle sur ce que l'on fait avec le keypad, et n'ayant pas pu trouver ça sur les librairies existantes, nous avons réutilisé un code que l'un de nous a eu l'occasion de coder lors de son semestre à Lisbonne, mais en langage C. Nous avons donc traduit le code en python mais par manque de temps, nous n'avons pas pu l'optimiser.

```python
import RPi.GPIO as GPIO
import time
import os
from Motor_Config import MotorControl
from PSJoystick import MyController



class Keypad:
    def __init__(self,motors_dico):

        self.MC=motors_dico

        Row1=14
        Row2=15
        Row3=18
        Row4=23

        Col1=5
        Col2=6
        Col3=13
        Col4=19

```

```python
        GPIO.setwarnings(False)
        GPIO.setmode(GPIO.BCM)

        GPIO.setup(Row1, GPIO.OUT)
        GPIO.setup(Row2, GPIO.OUT)
        GPIO.setup(Row3, GPIO.OUT)
        GPIO.setup(Row4, GPIO.OUT)

        GPIO.output(Row1, GPIO.HIGH)
        GPIO.output(Row2, GPIO.HIGH)
        GPIO.output(Row3, GPIO.HIGH)
        GPIO.output(Row4, GPIO.HIGH)

        GPIO.setup(Col1, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(Col2, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(Col3, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(Col4, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
```

Figures 11 et 12 : Code configuration et mappage du keypad (1 et 2)

Notre Keypad est notre contrôleur principal, il peut contrôler le moteur mais aussi passer la main à la manette. C'est pourquoi on importe la Class MyController et MotorControl.

Comme pour la manette à l'appelle du keypad nous lui rentrons en paramètre le dictionnaire des moteurs pour pouvoir contrôler les moteurs un par un. On le stock donc dans une variable de class self.MC. On associe les pins utilisés sur la Raspberry au variable Row et Col correspondant. Puis on initialise les pins des lignes en pin de sortie et on les met directement en état haut. On initialise ensuite les pins des colonnes en pins d'entrées et on les associe digitalement à des résistance pull-down, la Raspberry nous permet d'utiliser des résistances pull-up et pull-down en software. Les résistances pull-down permettent de maintenir les pins d'entrées en état bas lorsqu'aucune valeur n'est rentrée. Cela évite d'avoir des valeurs flottantes lorsqu'aucune touche n'est appuyée.

```
43          self.adresseRow=[Row1,Row2,Row3,Row4]
44          self.adresseCol=[Col1, Col2, Col3, Col4]
45          self.press_function= [self.key_1,self.key_4,self.key_7,self.key_etoile,
46                                self.key_2,self.key_5,self.key_8,self.key_0,
47                                self.key_3,self.key_6,self.key_9,self.key_diez,
48                                self.key_A,self.key_B,self.key_C,self.key_D]
49          self.unpress_function= [self.unkey_1,self.unkey_4,self.unkey_7,self.unkey_etoile,
50                                  self.unkey_2,self.unkey_5,self.unkey_8,self.unkey_0,
51                                  self.unkey_3,self.unkey_6,self.unkey_9,self.unkey_diez,
52                                  self.unkey_A,self.unkey_B,self.unkey_C,self.unkey_D]
53          self.nombreused= ['1','4','7','*',
54                            '2','5','8','0',
55                            '3','6','9','#',
56                            'A','B','C','D']
```

Figure 13 : Code configuration et mappage du keypad (3)

Nous créons ici différents tableaux. La première pour regrouper les adresses des lignes et la deuxième pour regrouper les adresses des colonnes.
Puis nous faisons 3 tableaux qui correspondent à nos matrices d'entrée. Les variables à l'intérieur vont être sélectionnées en fonction du bouton appuyé.

```
58      def listener(self):
59          col_number=0
60          row_number=0
61
62          print("c'est initialisé")
63
64          while col_number<4:
65              value=GPIO.input(self.adresseCol[0])
66              value2=GPIO.input(self.adresseCol[1])
67              value3=GPIO.input(self.adresseCol[2])
68              value4=GPIO.input(self.adresseCol[3])
69              which_col = [value, value2, value3, value4]
70
71              if which_col[col_number]!=0 :
72                  row_number=0
73                  time.sleep(0.1)
74                  while row_number<4 :
75                      GPIO.output(self.adresseRow[row_number], GPIO.LOW)
76
77                      which_col[col_number]= GPIO.input(self.adresseCol[col_number])
78                      if which_col[col_number]==0 :
79
80                          GPIO.output(self.adresseRow[0], GPIO.HIGH)
81                          GPIO.output(self.adresseRow[1], GPIO.HIGH)
82                          GPIO.output(self.adresseRow[2], GPIO.HIGH)
83                          GPIO.output(self.adresseRow[3], GPIO.HIGH)
84
85                          which_col[col_number]= GPIO.input(self.adresseCol[col_number])
86                          pressed_key= col_number*4 +row_number
87
88                          #self.press_function[pressed_key](self.nombreused[pressed_key])
89
90                          print("pressed :",self.nombreused[pressed_key])
91
92                          while(which_col[col_number]!=0):
93                              self.press_function[pressed_key](self.nombreused[pressed_key])
94                              which_col[col_number]= GPIO.input(self.adresseCol[col_number])
95
96                          self.unpress_function[pressed_key](self.nombreused[pressed_key])
97
98                          time.sleep(0.5)
99                          row_number=4
100                     row_number=row_number+1
101             if col_number==3 :
102                 col_number=0
103             else :
104                 col_number+=1
```

Figure 14 : Code configuration et mappage du keypad (4)

Le principe du keypad est expliqué dans la partie électronique. On a une variable col_number qui s'incrémente de 0 à 4 et qui tourne en boucle infinie car si cette variable vaut 4 elle est remise à 0. Cela sert à regarder la valeur de chaque colonne et que si une des colonnes (qui lit un état bas par défaut) lit un état haut, on essaye de savoir quelle ligne nous correspond.
On stocke donc la valeur que reçoivent les colonnes dans un tableau (l.80 à l.85), et si l'une des valeurs est différentes de 0, on sait que c'est un bouton de cette colonne qui est appuyé.

Maintenant pour vérifier la ligne du bouton appuyé, on fait incrémenter la variable row_number de 0 à 4. Pour chaque valeur de row_number on met à l'état bas la ligne correspondante et on vérifie si la colonne lit un changement d'état. Si la colonne lit un état bas, on a trouvé la colonne et la ligne correspondant au bouton que nous avons appuyé.
Dès que c'est trouver, on remet les pins des lignes en état haut ; à ce moment-là, la colonne lit un état haut et c'est seulement lorsque l'on relâche le bouton que le pin de la colonne lira un état bas (celui de la résistance pull-down).
Grâce à cette méthode nous pouvons lancer des fonctions que nous voulons lancer qu'une seule fois si le bouton est appuyé, relancer des fonctions tant que le bouton est appuyé, et lancer des fonctions lorsque c'est relâché. On peut donc utiliser la méthode d'incrémentation de la position des moteurs qui nécessite d'être appelé tant que le bouton est appuyé, et utiliser la méthode d'arrêt des moteurs lorsque l'on relâche le bouton.
On a un time.sleep d'une demie seconde à chaque appui de bouton pour laisser le temps aux états de se recalibrer.
La variable pressed_key nous fait le calcul pour avoir la position du bouton correspondant à la ligne et la colonne dans les matrices de fonctions.

```
110     #Controle de la Base
111     def key_4(self,var):
112         self.MC["motor_base"].move_min1()
113
114     def key_6(self,var):
115         self.MC["motor_base"].move_max1()
116
117     def unkey_4(self,var):
118         self.MC["motor_base"].stop_now()
119
120     def unkey_6(self,var):
121         self.MC["motor_base"].stop_now()
122
```

```
157     #Controle de la Pince
158     def key_0(self,var):
159         self.MC["motor_pince"].move_max1()
160
161     def key_diez(self,var):
162         self.MC["motor_pince"].move_min1()
163
164     def unkey_0(self,var):
165         self.MC["motor_pince"].stop_now()
166
167     def unkey_diez(self,var):
168         self.MC["motor_pince"].stop_now()
169
```

Figures 15 et 16 : Code configuration et mappage du keypad (5 et 6)

Key_4 est la méthode qui permet de mettre la position du moteur de la base à 0° et key_6 à 180°, en relâchant les boutons, les méthodes unkey_4 et unkey_6 se déclenchent et stoppent le moteur. On utilise donc le même principe pour la pince.
On utilise les boutons 4 et 6 pour faire tourner la base. Et les boutons 0 et # pour contrôler la pince.

```
123     #Controle Articulation 1
124     def key_8(self,var):
125         self.MC["motor_art_1"].move_min()
126
127     def key_2(self,var):
128         self.MC["motor_art_1"].move_max()
129
130     def unkey_8(self,var):
131         #self.MC["motor_art_1"].stop_now()
132         pass
133
134     def unkey_2(self,var):
135         #self.MC["motor_art_1"].stop_now()
136         pass
137
```

```
140     #Controle Articulation 2
141     def key_1(self,var):
142         self.MC["motor_art_2"].move_min()
143
144     def key_7(self,var):
145         self.MC["motor_art_2"].move_max()
146
147     def unkey_1(self,var):
148         #self.MC["motor_art_2"].stop_now()
149         pass
150
151     def unkey_7(self,var):
152         #self.MC["motor_art_2"].stop_now()
153         pass
154
```

Figures 17 et 18 : Code configuration et mappage du keypad (7 et 8)

Les moteurs des articulations ont à peu près le même principe que les 2 autres moteurs mais en relâchant les boutons on ne doit juste rien faire, on ne met pas la PWM à 0.
Et ici comme pour la manette de PS4, on utilise les méthodes move_min et move_max pour les moteurs pour modifier les valeurs de position et la maintenir.
On utilise ici les touches 2 et 8 pour l'articulation 1 et les touches 1 et 7 pour l'articulation 2.

```
181     #relache tout les moteurs
182     def key_5(self,var):
183         self.MC["motor_base"].stop_now()
184         self.MC["motor_art_1"].stop_now()
185         self.MC["motor_art_2"].stop_now()
186         self.MC["motor_pince"].stop_now()
187
```

Figure 19 : Code configuration et mappage du keypad (9)

Vu que le maintien d'une position via la PWM fait trembler les moteurs, en appuyant sur 5 on met toute les PWM à 0 même si cela relâche les moteurs. On a le même principe qu'un bouton de Kill program. Mais nous pouvons reprendre nos mouvements après.

```
171     #Connection a la manette
172     def key_etoile(self,var):
173         print("Bluetooth connection")
174         try :
175             os.system("bash /home/pi/connect.sh")
176             controller = MyController(interface="/dev/input/js0", connecting_using_ds4drv=True,motors_dico=self.MC)
177             controller.listen(timeout=10)
178         except:
179             print("no more Bluetooth connection")
```

Figure 20 : Code configuration et mappage du keypad (10)

La touche * nous permet de nous connecter à la manette de PS4, on essaye de lancer le script connect.sh, pour connecter la manette via Bluetooth et on lance la méthode listen de la manette pour l'utiliser, si la connexion ne s'est pas faite au bout de 10s, on repasse sur le keypad.

On ne peut pas utiliser les 2 contrôleurs simultanément (le Keypad et la manette de PS4). Donc quand on se connecte sur la manette de PS4, il faut la déconnecter pour repasser sur le keypad.

2.4 Organisation Main

```python
from Motor_Config import MotorControl
from Keypad import Keypad
from PSJoystick import MyController
import os

def init_all():
    base_motor=MotorControl(gpio_motor=17,initpos=30)
    art1_motor=MotorControl(gpio_motor=27)
    art2_motor=MotorControl(gpio_motor=22)
    pince_motor=MotorControl(gpio_motor=10,pos_max=12,pos_min=2,frequency=50)

    motor_dict= {
        "motor_base":base_motor,
        "motor_art_1":art1_motor,
        "motor_art_2":art2_motor,
        "motor_pince":pince_motor,
    }

    key=Keypad(motors_dico=motor_dict)
    key.listener()

    #controller = MyController(interface="/dev/input/js0", connecting_using_ds4drv=True,motors_dico=motor_dict)
    #controller.listen(timeout=10)


if __name__=="__main__":
    init_all()
```

Figure 21 : Code main

Ici on lance tout notre programme. On crée un objet de moteur pour chaque moteur en associant le pin pour chacun, leur position initiale s'ils en ont, la position minimale et maximale et la fréquence si ça doit changer.
On met tous ces objets dans le dictionnaire motor_dict, et on leur donne des noms que l'on réutilise dans nos Class Keypad et Controller (la manette de PS4).
Puis on lance le listener du keypad en mettant en paramètre de la classe le dictionnaire.

2.5 Automatisation

On utilise des scripts Bash pour automatiser notre projet, car nous voulons qu'il fonctionne sans utiliser l'interface graphique de la Raspberry Pi.

```
2  source /home/pi/Desktop/Rasp-Hand/HandProject/bin/activate
3
4  python /home/pi/Desktop/Rasp-Hand/Code/ProjectMain.py
```

Figure 22 : Automatisation (1)

Ce script est nommé runarm.sh, il permet d'activer notre environnement puis lancer le fichier Python ProjectMain.py qui est notre Main.
On lance ce script à l'allumage de la Raspberry Pi en rajoutant "sudo bash /home/pi/runarm.sh &" dans le rc.local qui est un script qui se lance quand on allume la Raspberry Pi.

```
2  bluetoothctl << EOF
3  connect 40:1B:5F:E3:56:CC
4  EOF
```

Figure 23 : Automatisation (2)

Voici le script connect.sh qui permet de connecter un appareil en Bluetooth avec une adresse MAC donnée. Ici, celle de la manette de la PS4.

On a le même script pour déconnecter la manette.

3. Électronique

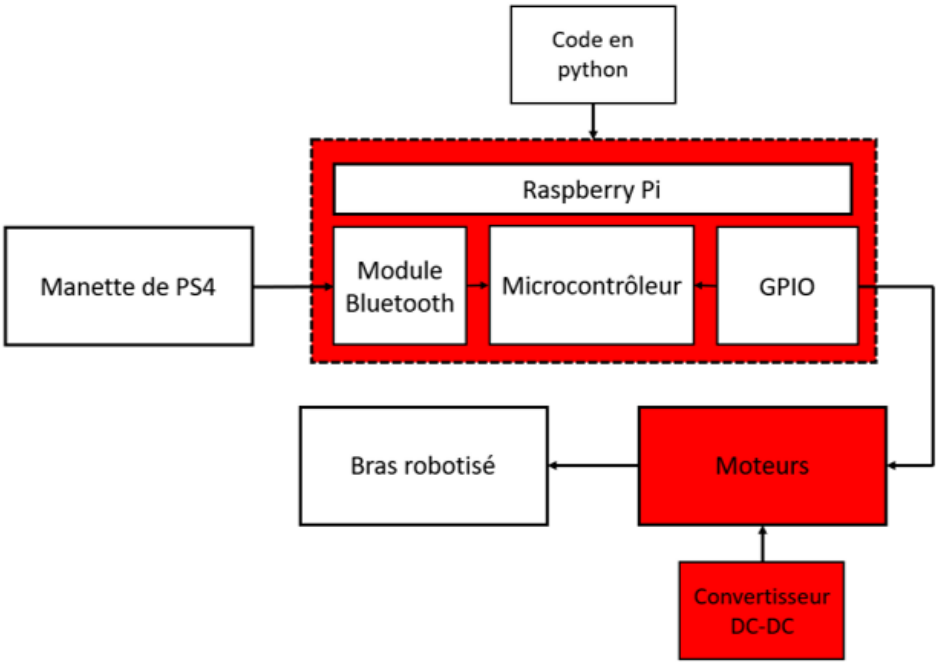


Figure 24 : Schéma global du projet (partie électronique)

3.1 Composants utilisés

- *Raspberry Pi 3B +*

Figure25 : Raspberry Pi 3B+

Le Raspberry Pi est un nano-ordinateur monocarte à processeur ARM de la taille d'une carte de crédit conçu par des professeurs du département informatique de l'université de Cambridge dans le cadre de la fondation Raspberry Pi3.

Le 14 mars 2018, la fondation Raspberry Pi annonce la mise à jour du Raspberry Pi 3 vers le modèle B+. On y trouve une mise à jour du processeur Broadcom BCM2837B0 64 bit à quatre cœurs ARM Cortex-A53 cadencé à 1,4 GHz au lieu du 1,2 GHz. La puce Cypress CYW43438 est remplacée par une nouvelle puce CYW43455 supportant le WiFi Dual-band 802.11ac et la version 4.2 du Bluetooth. Et d'une prise en charge du Power over Ethernet grâce à un élément supplémentaire.

- *Servomoteurs MG996R*

Figure 26 : Servomoteurs MG996R

Le MG996R est un servomoteur numérique robuste à engrenages métalliques. Il dispose d'un couple de 11kg/cm et est très utilisé dans les domaines de la robotique, modélisme, domotique et prototypage.
Comme les autres servomoteurs RC, le moteur tourne de 0° à 180° en fonction du cycle d'utilisation de la PWM fournie à sa broche de signal.
Un signal de 0.5ms correspond à 0° et un signal de 2.5ms à 180°. Le temps correspondant à une fréquence de 50Hz est 20 ms. On divise donc 0.5 par 20 pour avoir le dutyCycle correspondant
(notre pos_min et pos_max de l'initialisation du moteur).

Câble rouge : + 5V
Câble orange : PWM
Câble marron : GND

- *Convertisseur DC-DC LM2596*

Figure 27 : Convertisseur DC-DC LM2596

Un convertisseur DC-DC permet de délivrer une tension différente de celle qu'il reçoit.
Pour notre projet, nous utilisons un convertisseur DC-DC LM2596 pouvant recevoir entre 3,2V et 40V et transmettre en sortie de 1,25V à 35V.
Celui-ci est essentiellement composé de deux condensateurs (un en entrée et l'autre en sortie), un transistor, des diodes, une bobine et un potentiomètre. On peut observer la tension de sortie grâce à un voltmètre et ainsi régler la tension souhaitée. Pour modifier cette tension, il suffit de tourner la vis sur le potentiomètre.
Ici, nous désirons une tension de 5V pour alimenter nos moteurs.

- *Clavier 4x4 (keypad)*

Figure 28 : Clavier 4x4 (keypad)

Le clavier possède 4 broches pour les colonnes et 4 broches pour les lignes qui se relient au moment où l'on appuie sur un bouton. On met donc les colonnes en entrées avec une résistance pull down qui permet de lire un état bas quand rien d'autre n'est entré cela évite de recevoir une valeur flottante et ainsi fausser nos résultats et les lignes en sorties.
Les lignes envoient une valeur différente de 1 (l'état haut) et au moment où l'on appuie sur une touche, la colonne correspondante lit une valeur et après il faut donc déterminer la lignes du bouton appuyé. Pour cela, sachant que la colonne lit un état haut car un bouton est appuyé, on va mettre à l'état bas une par une les lignes afin que la colonne lise un état bas de nouveau. Dès qu'on connait la ligne et la colonne correspondant à la touche appuyée, on sait quelle touche est appuyé.

- *Manette de PS4*

Instructions pour les servomoteurs

Analogue gauche : Avancer l'articulation 1
Analogue droit : Reculer l'articulation 1
Carrée : Tourner à gauche la base
Rond : Tourner à droite la base
R1 : Serrer la pince
L1 : Desserrer la pince
Option : Passer de la manette au keypad

Figure 29 : Manette de PS4

3.2 Schéma électronique

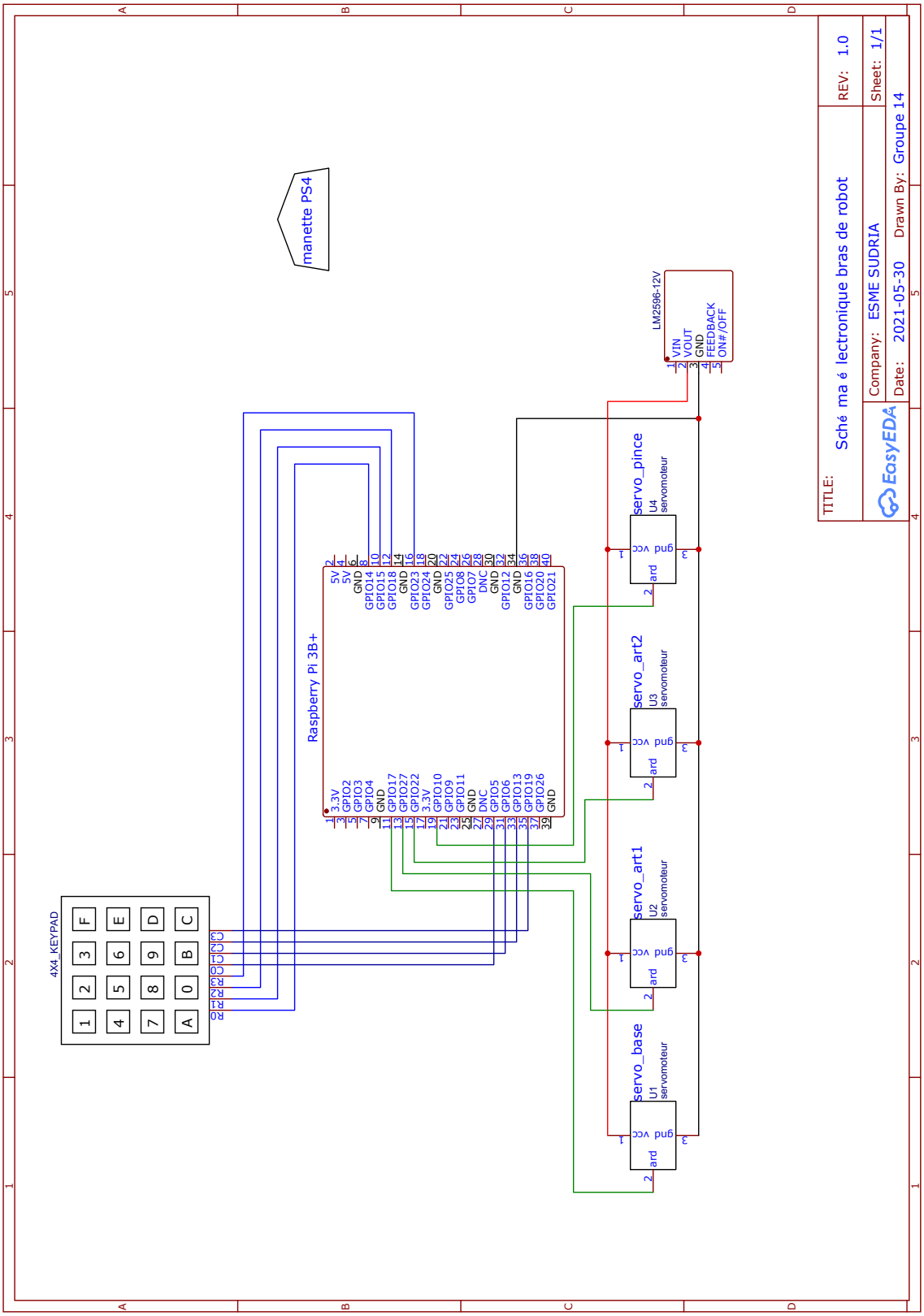


Figure 30 : Schéma électronique

4. Mécanique

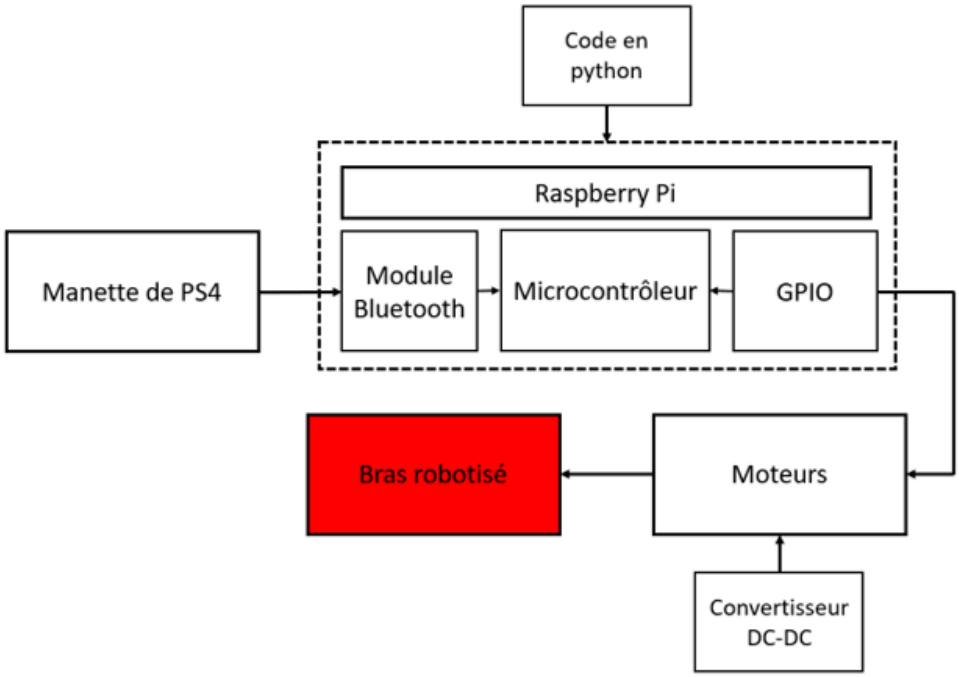


Figure 31 : Schéma global du projet (partie mécanique)

4.1 Calculs théoriques

Dans ce rapport, nous avons souhaité effectuer les calculs vus en cours de Systèmes Articulés. Il nous semblait important de faire cette partie théorique, afin de mieux intégrer la pratique.

Représentation du bras robotisé

Notre bras robotisé évolue dans l'espace selon trois axes, comme annoncé plus haut : x, y et z. Nos articulations sont toutes rotoïdes, pivotant selon les axes z0, z1 et z2, avec des angles respectifs theta1, theta2 et theta3.
Nous avons également associé d1 à la hauteur du tronc, a2 à la longueur du bras 1, et a3 à la longueur du bras 2 plus celle de la pince.

Pour cette première partie, nous avons représenté notre robot R-R-R suivant :

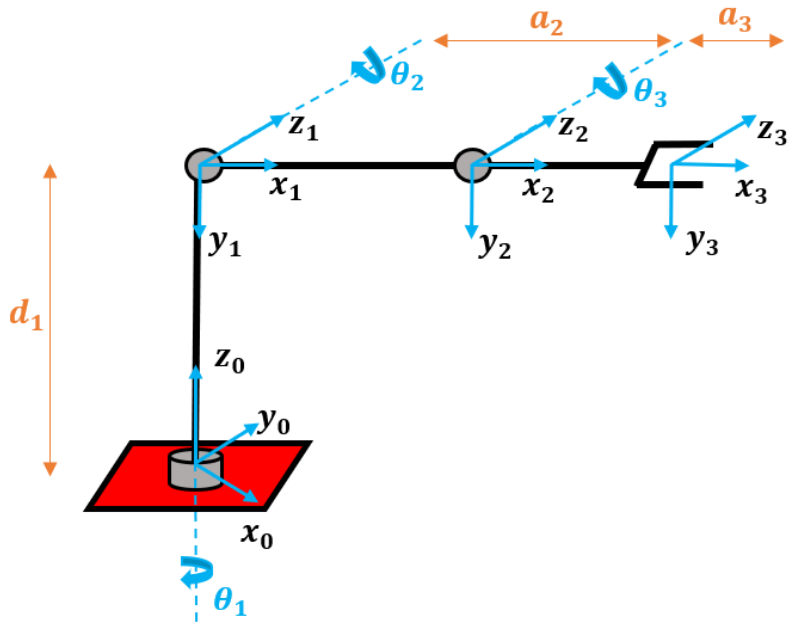


Figure 32 : La D-H représentation du bras de robot

Paramètres de Denavit-Hartenberg

Nous remplissons le tableau ci-dessous pour chaque paramètre, ce qui va nous aider pour la suite des calculs.

Axe	θ	d	a	α
1	θ1	d1	0	-π/2
2	θ2	0	a2	0
3	θ3	0	a3	0

Modèle géométrique direct

Le modèle géométrique direct permet de calculer les coordonnées opérationnelles du robot avec sa position et ses coordonnées articulaires.

Ensuite, nous calculons le modèle géométrique direct. Pour cela, nous devons trouver :

$$T_{base}^{outil} = T_0^3 = T_0^1 . T_1^2 . T_2^3$$

$$T_0^3 = \begin{bmatrix} C_1 & 0 & -S_1 & 0 \\ S_1 & 0 & C_1 & 0 \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} C_2 & -S_2 & 0 & a_2C_2 \\ S_2 & C_2 & 0 & a_2S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} C_3 & -S_3 & 0 & a_3C_3 \\ S_3 & C_3 & 0 & a_3S_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\Longrightarrow T_{base}^{outil} = T_0^3 = \begin{bmatrix} C_1C_{23} & -C_1S_{23} & S_1 & C_1(a_3C_{23} + a_2C_2) \\ S_1C_{23} & -S_1S_{23} & C_1 & S_1(a_3C_{23} + a_2C_2) \\ -S_{2-3} & -C_{2-3} & 0 & -a_3S_{23} - a_2S_2 + d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Modèle géométrique inverse

Le modèle géométrique inverse permet de déterminer la configuration des liaisons, en fonction de la position et de l'orientation de l'effecteur du robot.

Nous souhaitons ensuite trouver le modèle géométrique inverse. Nous devons trouver w(q) :

$$w(q) = \begin{bmatrix} C_1(a_3C_{23} + a_2C_2) \\ S_1(a_3C_{23} + a_2C_2) \\ -a_3S_{23} - a_2S_2 + d_1 \\ S_1 \exp\left(\frac{q_3}{\pi}\right) \\ C_1 \exp\left(\frac{q_3}{\pi}\right) \\ 0 \end{bmatrix}$$

On cherche ensuite à exprimer q_1, q_2, q_3.
Pour cela nous utilisons plusieurs formules avec lesquelles on trouve q_1 et q_3 assez facilement :

$$q_1 = \operatorname{atan2}(w_2, w_3)$$

$$q_3 = \pi . \ln \sqrt{w_4^2, w_5^2}$$

Pour trouver q_2 , on isole C_2 et S_2 afin de l'exprimer en fonction de w_1 , w_2 et w_3 :

$$C_2 = \frac{w_1 - C_1 a_3 - C_{23}}{C_1 a_2}$$

$$S_2 = \frac{w_3 + a_3 S_{23} - d_1}{a_2}$$

$$q_2 = \operatorname{atan2}\left(\frac{w_3 + a_3 S_{23} - d_1}{a_2}, \frac{w_1 - C_1 a_3 - C_{23}}{C_1 a_2}\right)$$

Matrice Jacobienne de l'outil

Cette matrice sert à exprimer la vitesse de l'outil en fonction des vitesses articulaires. Elle est utile lorsqu'on souhaite commander un robot pour qu'il suive la trajectoire au préalable calculée.
Pour déterminer $v(q)$, on dérive w(q) par apport à q_1, q_2 et q_3. On obtient :

$$v(q) = \begin{bmatrix} -S_1(a_3 C_{23} + a_2 C_2) & -C_1(a_3 S_{23} + a_2 S_2) & -a_3 C_1 S_{23} \\ C_1(a_3 C_{23} + a_2 C_2) & -S_1(a_3 S_{23} + a_2 S_2) & -a_3 S_1 S_{23} \\ 0 & -a_3 C_{23} - a_2 C_2 & -a_3 C_{23} \\ C_1 exp\left(\frac{q_3}{\pi}\right) & 0 & 0 \\ -S_1 exp\left(\frac{q_3}{\pi}\right) & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Matrice Jacobienne du manipulateur

$$J(q) = \begin{bmatrix} -S_1(a_3 C_{23} + a_2 C_2) & -C_1(a_3 S_{23} + a_2 S_2) & -a_3 C_1 S_{23} \\ C_1(a_3 C_{23} + a_2 C_2) & -S_1(a_3 S_{23} + a_2 S_2) & -a_3 S_1 S_{23} \\ 0 & -a_3 C_{23} - a_2 C_2 & -a_3 C_{23} \\ 0 & -S_1 & S_1 \\ 0 & C_1 & C_1 \\ 1 & 0 & 0 \end{bmatrix}$$

4.2 Procédé

En ce qui concerne la partie mécanique de notre projet, nous avons longuement hésité sur la manière de procéder.

Notre première idée était de construire l'entièreté du robot avec des pièces de mécano dans le but d'obtenir un bras de robot le plus solide possible.
Finalement, nous nous sommes rendu compte que cela nous compliquerait la tâche, surtout pour assembler les servomoteurs aux différentes parties du robot. Et puis, nous ne trouvions pas exactement ce que nous avions en tête sur le marché.
Nous avons donc décidé de partir sur une création totale des pièces sur le logiciel de modélisation 3D SolidWorks. Avant de commencer, nous avons établis un schéma sur papier avec les mesures que nous voulions donner à notre robot, sa forme globale et les liaisons que nous allions mettre en place entre chaque partie. Il se compose d'une base, d'un tronc surélevé permettant de faire pivoter le robot, de deux bras, et d'une pince. Afin de ne pas mettre tout le poids du robot sur les moteurs, nous avons pensé à utiliser des roulements à billes. Pour pouvoir avoir une vue d'ensemble pour l'assemblage sur SolidWorks et que cela soit plus simple pour les mesures, nous avons trouvé une modélisation 3D des moteurs MG996R que nous allions utiliser, et nous avons modélisé leurs hélices ainsi que les roulements à billes.

Finalement, nous avons décidé de fixer notre robot sur une planche en bois pour qu'il puisse gagner en stabilité, et si besoin, d'utiliser les pièces de mécano pour le renforcer.

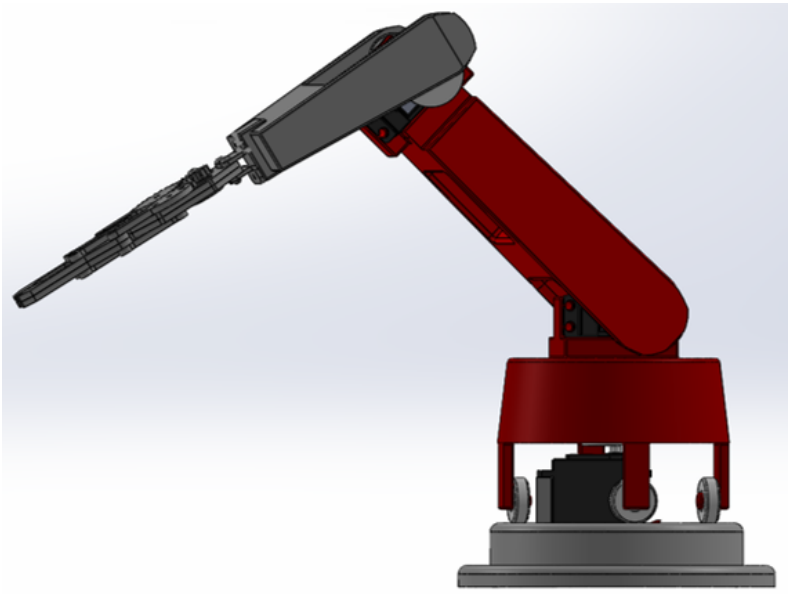


Figure 15 : Modélisation 3D du bras de robot (Solidworks)

4.2.1 Base

Pour la première pièce qui est la base, nous avons simplement créé un socle de 15cm de diamètre, pouvant contenir un premier moteur dans un creux. Ce premier moteur permet au bras robotisé de pivoter sur lui-même. Nous avons également fait des trous pour pouvoir fixer le moteur à la base et fixer celle-ci sur une planche en bois, ainsi que des rails pouvant accueillir les câbles.

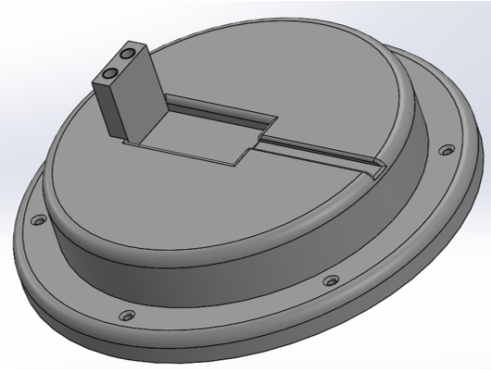


Figure 16 : Première pièce : la base

4.2.2 Tronc

La deuxième pièce est ce que nous avons appelé le “tronc”. Il s’agit d’une pièce cylindrique de 12cm, venant se fixer au moteur à l’aide d’un trou creusé sur le dessous, et venant se poser sur la base avec quatre roulements à billes.

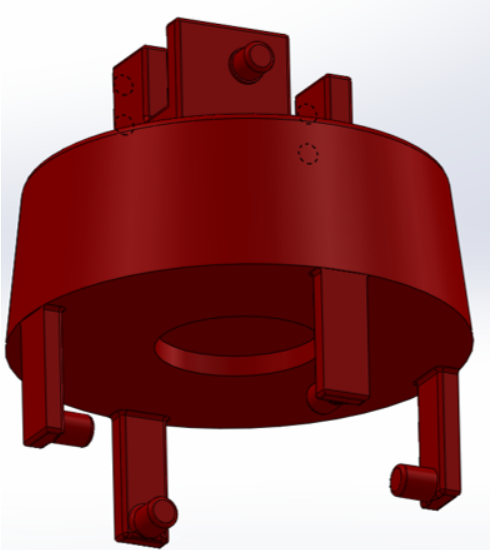


Figure 17 : Seconde pièce : le tronc (vue du dessous)

Sur le dessus de la pièce, nous avons construit un espace qui accueillera le deuxième moteur. Celui-ci va permettre au bras de bouger en hauteur. Nous avons également fait un axe permettant de fixer le premier bras à l'aide d'un roulement à billes.

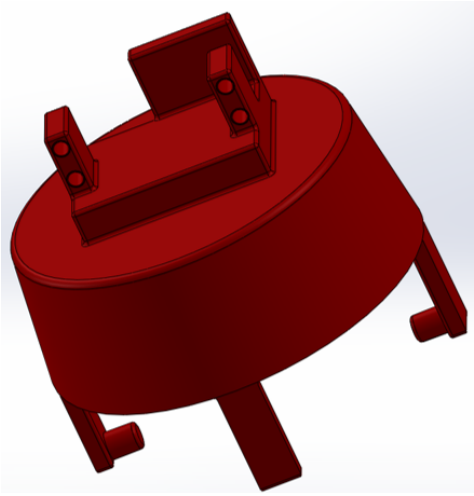


Figure 18 : Seconde pièce : le tronc (vue du dessus)

4.2.3 Articulation 1

La troisième pièce est le premier bras de notre robot. Il mesure 20cm de hauteur et 7cm de largeur. Le troisième moteur sera positionné en haut du bras et il permettra au deuxième bras de bouger en hauteur et d'apporter également la longueur au bras robotisé. Nous avons créé un espace sur une des pattes du bras afin d'y glisser un roulement à billes. Celui-ci fera la liaison avec le tronc.

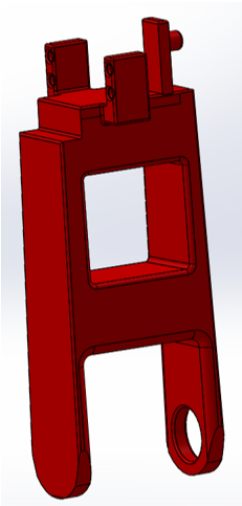


Figure 19 : Troisième pièce : l'articulation 1

4.2.4 Articulation 2

Ensuite, il y a le deuxième bras qui mesure 11,70 cm. Nous mettrons également un roulement à billes pour pouvoir fixer cette pièce au premier bras.

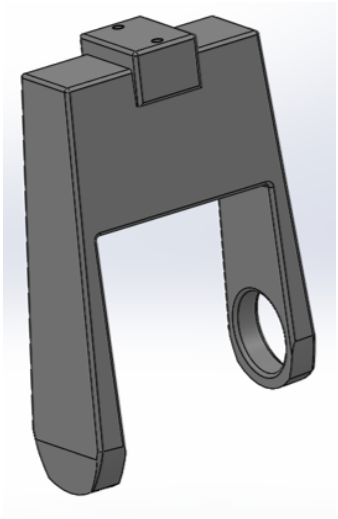


Figure 20 : Quatrième pièce : l'articulation 2

4.2.5 Pince

La cinquième pièce est la pince, pièce finale du robot, qui mesure 13cm de long. Le quatrième moteur va être fixé en dessous de la pince à l'aide d'une vis, et entraîner un engrenage central qui entraînera les deux engrenages secondaires qui, eux-mêmes, feront bouger les bras de la pince. Afin que la pince attrape mieux les objets, nous avons fixé des bouts de caoutchouc sur ses extrémités.

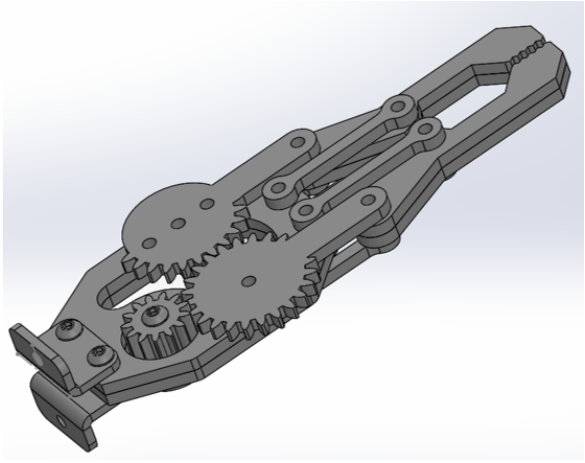


Figure 21 : Cinquième pièce : la pince

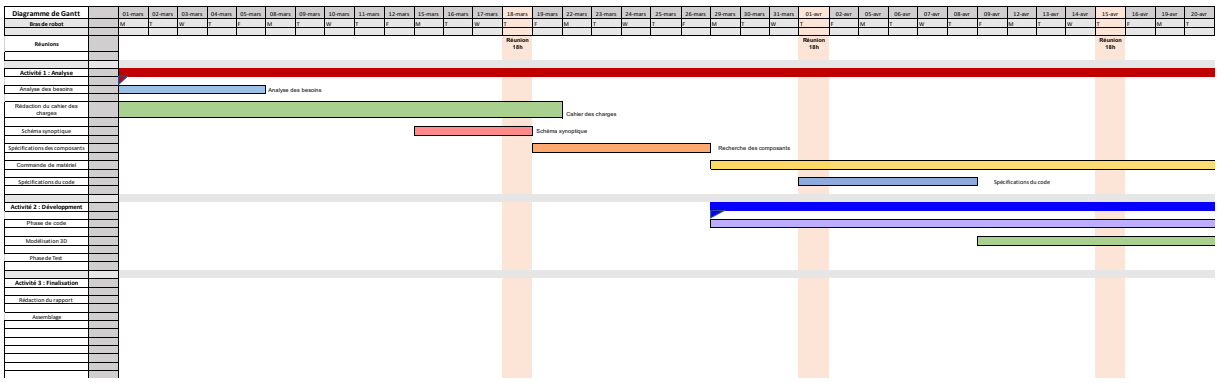
5. Management

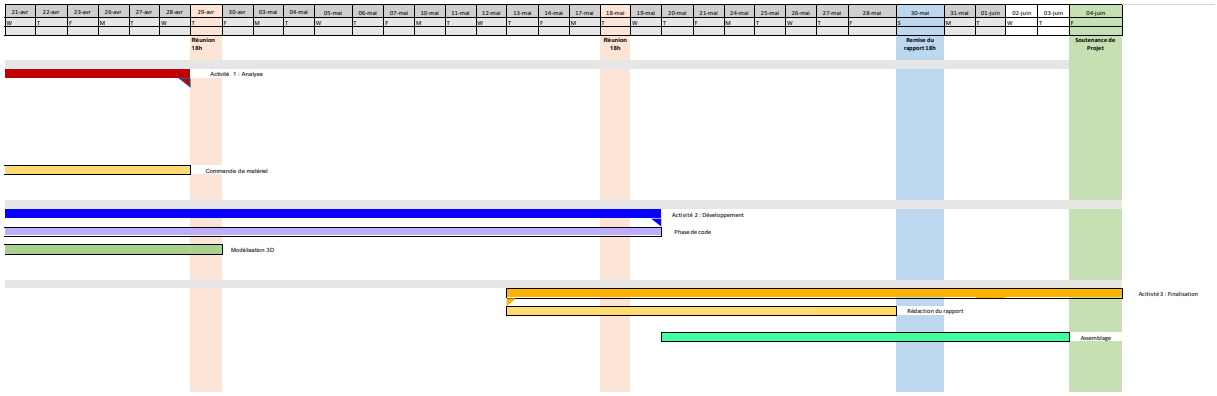
5.1 Diagramme de Gantt

Le diagramme de Gantt, couramment utilisé en gestion de projet, est l'un des outils les plus efficaces pour représenter visuellement l'état d'avancement des différentes activités (tâches) qui constituent un projet.

Il permet de répertorier toutes les tâches à accomplir pour mener le projet à bien, et indique la date à laquelle ces tâches doivent être effectuées (le planning).

Le diagramme de Gantt étant trop grand, nous l'avons coupé en deux pour qu'il puisse être plus lisible.





5.2 Difficultés et Résultats

Au cours de la réalisation de nos travaux, nous avons rencontré diverses difficultés dans différents domaines.

Modélisation/Impression 3D

- Manque de précision au niveau de la modélisation du bras n°1 et 2, notamment lors de l'emboîtement de ces deux pièces. Il a donc fallu réimprimer ces deux pièces en modifiant certaines mesures sur Solidworks.
- Lors de la mise en place de supports (utiles lors de l'impression), certains d'entre eux étaient inutiles et difficiles à retirer de la pièce.

Composants

- Nous avons eu 4 types de servomoteurs à tester ; l'un d'entre eux étaient défectueux, et les deux autres ne correspondaient pas au cahier des charges qui nous a été donné.

Assemblage des pièces

- Par le manque de temps au niveau des impressions 3D, nous avons dû creuser approximativement le "tronc" pour y fixer le servomoteur qui le relie à la base.

Mise à part ces quelques difficultés, ce projet a été une réussite.
Tout d'abord, il a été accompli dans la limite du temps imparti et nous a permis d'acquérir de nouvelles compétences notamment au niveau de la modélisation 3D et du code.
De plus, la communication entre les membres de ce groupe et l'encadrant de notre projet était excellente même si cela s'est fait à distance via teams.
Nous sommes heureux de pouvoir présenter un projet fonctionnel qui peut venir en aide aux personnes à mobilité réduite.
Nous partageons l'entièreté du code ainsi que la documentation sur Github afin de permettre à n'importe qui de comprendre et même de réaliser ce projet.

6. Bilan

6.1 Conclusion

Ce travail nous a permis de concevoir un projet de A à Z : un bras robotisé bougeant sur trois axes et permettant d'attraper un objet pas fragile ni trop lourd, contrôlé par une manette de PS4. Nous sommes parvenus à le réaliser en utilisant nos connaissances en termes d'informatique, d'électronique et de mécanique. Nous en avons aussi beaucoup appris sur ces trois modules, et ce travail nous a permis de gagner en autonomie.

Nous avons également dû nous organiser en groupe : nous nous sommes réunis régulièrement à l'école ou via teams pour se répartir au mieux les tâches et avancer sur le projet petit à petit. Nous avons eu quelques soucis qui ont retardé le planning que nous nous étions imposés, mais nous avons réussi à trouver des solutions, et surtout à ne pas perdre espoir.

6.2 Ouverture

A l'avenir, nous pourrions envisager de créer un bras robotisé plus complexe que celui-ci : jouer sur les articulations comme rajouter un bras ou alors donner une fonction de rotation à la pince, créer une interface via laquelle on pourrait directement contrôler le robot, ajouter une caméra, mais également rendre plus « propre » le bras en créant une boîte pour cacher les composants électroniques et améliorer la modélisation 3D de nos pièces…

Il y a beaucoup de choses que l'on pourrait ajouter/améliorer, et cela permettrait une aide encore plus précieuse aux personnes à mobilité réduite qui souhaiteraient bénéficier d'un tel robot !

Sources

https://www.accessiblepourmoi.com/ce-que-dit-la-loi/les-chiffres-du-handicap/

https://www.monde-proprete.com/sites/default/files/familles_de_deficiences.pdf

https://objectifadvf.wordpress.com/2016/09/02/hemiplegie-paraplegie-tetraplegie/

https://poujouly.net/2012/05/22/convertisseur-dcdc/

https://www.solaris-store.com/content/48-principe-de-fonctionnement-d-un-convertisseur-dc-dc

https://fr.wikipedia.org/wiki/Raspberry_Pi#Modèle_3_B+_(Raspberry_Pi_3+)

https://components101.com/motors/mg996r-servo-motor-datasheet

https://raspberry-pi.fr/servomoteur-raspberry-pi/

Image Raspberry Pi 3B+
https://upload.wikimedia.org/wikipedia/commons/thumb/d/d2/Raspberry_Pi_3_Model_B.JPG/220px-Raspberry_Pi_3_Model_B.JPG

Image Convertisseur DC-DC LM2596
https://tse4.mm.bing.net/th?id=OIP.5nQd_U2SgdZRXKjP8jbKDAHaHa&pid=Api&P=0&w=300&h=300

Image Clavier 4x4 (keypad)
https://tse2.mm.bing.net/th?id=OIP.WIb3wZBgXtMjG8csV2ykeAAAAA&pid=Api&P=0&w=300&h=300

Image manette de PS4
https://tse4.mm.bing.net/th?id=OIP.EGTTn8Wsas8JAv8ocU4dvwAAAA&pid=Api&P=0&w=225&h=151