

INSTITUT SUPÉRIEUR
D'ÉLECTRONIQUE DE PARIS

TIPE

Bras mécanique et sudoku

					3			
5			7	1		2		
	4			9		8	6	
		3						8
	5	2	6			9	3	
9		7		3			5	
			3		5			
6				2		1		
	9		1				4	

Laurent Tainturier & Alphonse Terrier

supervisé par

M. Patrick COUVEZ

2016-2017

Table des matières

Introduction	2
1 Présentation et faits sur le sudoku	4
2 Électronique	5
2.1 Moteurs pas-à-pas	5
2.2 Schéma électrique	5
2.3 Premiers montages	5
2.4 Circuit imprimé	5
2.5 Écran LCD	6
3 Mécanique	8
3.1 Conception du chariot en 3D	8
4 Informatique	9
4.1 Présentation globale	9
4.2 Reconnaissance du sudoku	9
4.3 Résolution du sudoku	12
4.4 Interface graphique	12
4.5 Contrôle des moteurs et écriture	14
4.5.1 Moteurs pas-à-pas	14
4.5.2 Coordonnées polaires	14
4.5.3 Écriture des digits	14
A Fichier principal	18
B Script de résolution des sudokus	20
C Script de gestion de la caméra	24
D Script d’affichage du sudoku	25
E Script de gestion des moteurs pas-à-pas	30
F Script permettant l’écriture d’un sudoku	35
Mises en plan des pièces conçues en 3D	40

Introduction

Chapitre 1

Présentation et faits sur le sudoku

Le sudoku est un jeu sous forme de grille inspiré du carré latin et défini en 1979 par Howard Garns.

Cette grille est carrée et est divisée en n^2 régions de n^2 cases. Elle possède ainsi n^2 colonnes, n^2 lignes et n^4 cases. Dans la version la plus commune : $n = 3$.

Une grille de sudoku est préremplie, le but du jeu étant de la compléter selon la règle suivante :

- Chaque ligne, chaque colonne et chaque région doit contenir au moins une fois tous les de 1 à n^2 .

Une grille est considérée comme sudoku seulement si sa solution est unique.

Le minimum de cases remplies au préalable pour espérer que la dite solution soit bien unique est de 17, cela a été prouvé par une équipe islandais en 2012.

Chapitre 2

Électronique

2.1 Moteurs pas-à-pas

Nous avons utilisés dans ce projet deux moteurs pas-à-pas qui présentaient, par rapport à d'autres types de moteurs, les avantages suivants :

- une précision bien supérieure à celle de moteurs à courant continu ;
- un couple bien plus important que celui de servomoteurs ;
- mis sous tension, un déplacement fortuit du moteur n'est pas possible.

Les moteurs pas-à-pas sont notamment utilisés dans les systèmes nécessitant une grande précision comme les imprimantes 3D dans lesquelles ils sont très largement employés.

2.2 Schéma électrique

Le schéma ci-joint représente les principales connections au sein de notre montage, même s'il ne présente ni l'écran LCD, ni le détail de l'alimentation (notamment du convertisseur de tension).

Les moteurs pas-à-pas sont constitués de deux bobines qui sont reliées à deux drivers l293D . Ceux-ci sont pilotés par quatre sorties GPIO qui envoient des impulsions au driver qui alimente les bobines en 12V à tour de rôle, ce qui permet à la fois de contrôler le sens et la vitesse de de rotation des moteurs, en choisissant à quelle fréquence envoyer les impulsions.

2.3 Premiers montages

Nous avons réalisés nos premiers montages avec une breadboard facilitant les tests. Une simulation du schéma a également été réalisée sous *Fritzing*

2.4 Circuit imprimé

Pour rendre notre bras mécanique plus compact et ainsi rentrer dans le cadre de l'optimalité, nous avons souhaité remplacer la breadboard par une

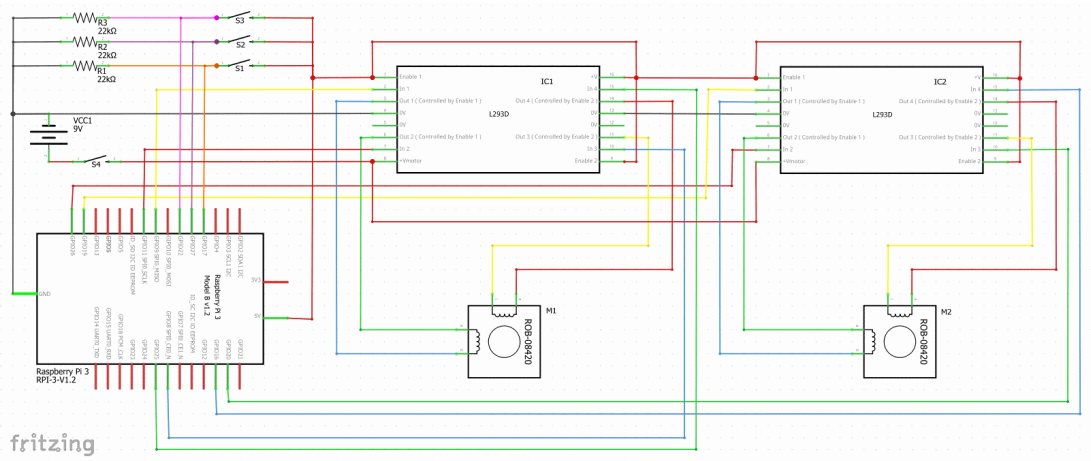


FIGURE 2.1 – Schéma électrique réalisé avec *Fritzing*

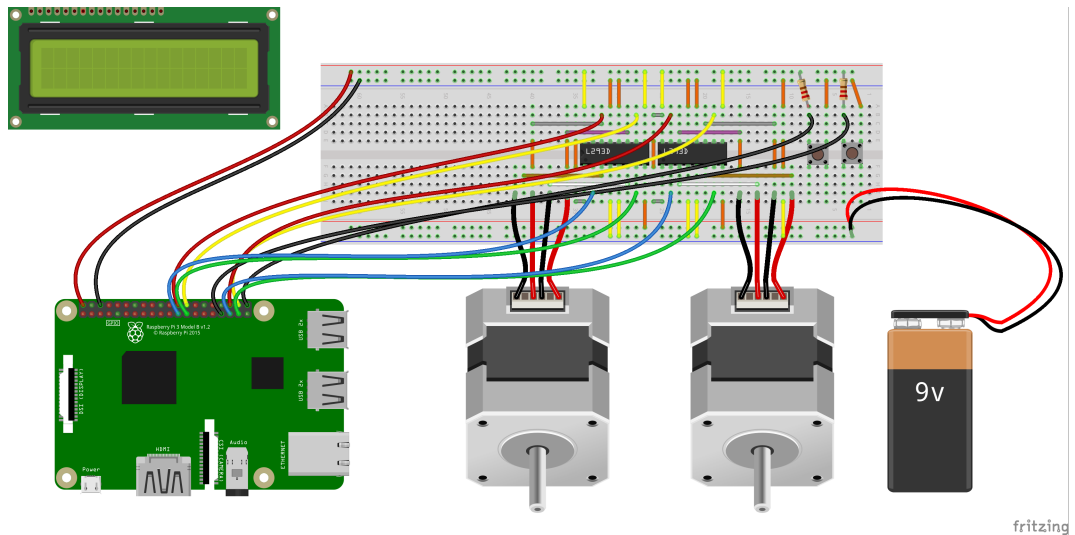


FIGURE 2.2 – Premier montage

solution bien plus compacte, à savoir un circuit imprimé. Celui-ci sera enficher directement sur les ports GPIO de la Raspberry. Nous avons de nouveau utilisé pour le réaliser le logiciel *Fritzing*, permettant la réalisation du circuit sur deux couches (symbolisées par les couleurs orange et jaune), permettant un cablage plus facile, car permettant les croisements. Nous avons utilisé *Fritzing* car il été assez facile de faire faire fabriquer le circuit imprimé pour une dizaine d'euros. Nous avons hésité à réaliser entièrement ce circuit par nous même, mais du fait de la présence de deux couches, cela se serait révéler très difficile.

2.5 Écran LCD

Pour rendre le bras mécanique autonome, nous avons intégré un écran LCD permettant à l'utilisateur de se repérer dans l'évolution des différents scripts sans disposer nécessairement d'un ordinateur ou d'un écran à proximité.

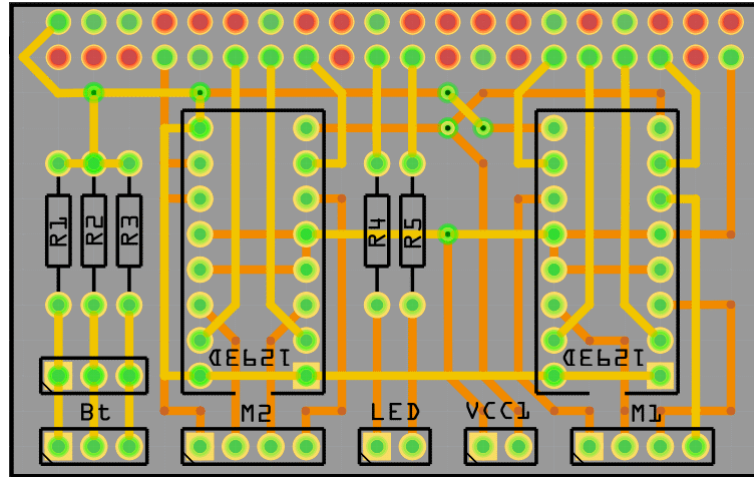


FIGURE 2.3 – Circuit imprimé réalisé avec *Fritzing*



FIGURE 2.4 – Écran LCD affichant le message de bienvenue

Chapitre 3

Mécanique

La premier défi qui s'est imposé à nous a été le choix d'une structure adéquate : solide et pratique. Notre choix s'est alors porté sur les Makerbeam. MakerBeam est un système de construction Open-Source basé sur des profilés ALU en T.

3.1 Conception du chariot en 3D

Chapitre 4

Informatique

4.1 Présentation globale

Tous les algorithmes développés dans le cadre de ce projet sont disponibles en annexe. Ils ont été, pour la plupart, développés en Python 3, les autres se basant sur Python 2 car certaines bibliothèques dont nous avons besoin, notamment pour la reconnaissance, n'étaient disponibles que sous Python 2.

Nous avons développé une programmation modulaire, permettant de travailler simultanément sur le projet, sans pour autant poser de problème de logistique. Ainsi nous avons dissocié tous les scripts ; que ce soit la reconnaissance, la résolution, l'affichage, la gestion de la caméra ou des servo-moteurs, etc. Pour cela, nous avons développé une relation qualifiable de maître-esclave entre nos scripts. Chacun des scripts dépend d'un fichier principal, appelé *main*, qui récupère les informations des scripts et donne les ordres adéquats à ceux-ci, selon la situation. Ainsi, les scripts ne sont pas reliés les uns aux autres mais seulement à ce script principal, ce qui permet d'ajouter ou d'enlever très facilement tel ou tel script, sans pour autant altérer le fonctionnement de l'ensemble, ce qui permet de tester chacun des scripts très facilement.

4.2 Reconnaissance du sudoku

Le script de reconnaissance du sudoku a été réalisé sous Python 2 avec le module de traitements d'image OpenCV. Il a été réalisé pour :

1. Reconnaître les chiffres dans une grille du sudoku
2. Déterminer la position spatiale de la grille

On photographie la grille avec une caméra Raspberry Pi (V2) comme celle-ci :

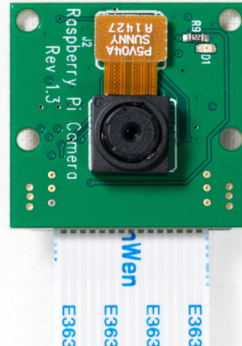


FIGURE 4.1 – Caméra Raspberry Pi V2

Voici la grille de sudoku qui nous servira d'exemple pour montrer toutes les actions du script :

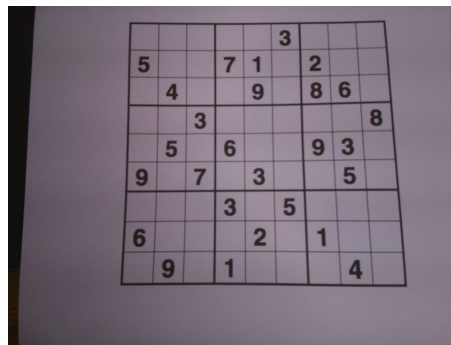


FIGURE 4.2 – Exemple de grille de sudoku

On applique sur cette photographie un filtre de type seuil (en anglais "threshold") qui va ensuite nous permettre de détecter les contours de la grille :

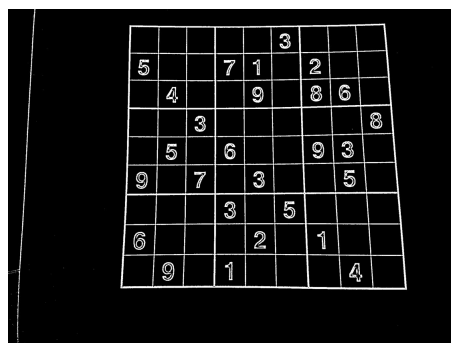


FIGURE 4.3 – Exemple de grille "seuillée"

Par cette transformation, on peut ensuite déterminer des équations de droites des contours extérieurs de la grille. Les coordonnées des intersections des droites seront celle des coins de la grille.

On découpe alors la grille de la photographie initiale en supprimant les éventuels effets de perspective.

					3			
5			7	1		2		
	4			9		8	6	
		3						8
	5		6			9	3	
9		7		3			5	
			3		5			
6				2		1		
	9		1				4	

FIGURE 4.4 – Exemple de grille découpée sans perspective

On découpe chaque petite case de la grille comme ci-dessous :

					3			
5			7	1		2		
	4			9		8	6	
		3						8
	5		6			9	3	
9		7		3			5	
			3		5			
6				2		1		
	9		1				4	

FIGURE 4.5 – Exemple de grille où chaque chiffre a été découpé

On utilise ensuite un module de reconnaissance de digits pour détecter les chiffres et on obtient la grille suivante, prête à être résolue :

					3			
5			7	1		2		
	4			9		8	6	
		3						8
	5	2	6			9	3	
9		7		3			5	
			3		5			
6				2		1		
	9		1				4	

FIGURE 4.6 – Exemple de grille prête à être résolue

4.3 Résolution du sudoku

Pour résoudre une grille de sudoku, un joueur utilise différentes méthodes de résolution, qui sont purement algorithmiques. Il utilise en grande majorité deux principales méthodes qui peuvent permettre de résoudre une grande partie des grilles disponibles sur le marché. On appelle ces deux méthodes *inclusion* et *exclusion*. Cependant les grilles de niveau supérieur font appel à des méthodes plus subtiles et souvent plus difficiles à appliquer en pratique qui se basent généralement sur des *paires* ou des *triplets*. Enfin, il existe une méthode, très difficilement utilisable par un joueur, appelée *backtracking* (ou *retour sur trace*), qui permet de résoudre toute grille de sudoku, même si celle-ci possède plusieurs solutions.

Inclusion

Exclusion

Paires Cette méthode possède différentes variantes :

-
-

La méthode des triplets est une généralisation de cette méthode à trois cases et non seulement deux.

Backtracking Cette méthode consiste tout d'abord à lister pour chacune des cases vierges les chiffres qui pourraient correspondre. On part d'une case vierge quelconque qu'on remplit par un des chiffres qu'on pourrait théoriquement placer et on liste de nouveau les possibilités des autres cases en fonction du chiffre précédemment ajouté puis on passe à une autre case vierge. S'il n'y a plus aucune possibilités qui pourraient correspondre, on revient sur nos pas et on change le chiffre qu'on venait d'insérer en une autre possibilité. S'il n'y a plus de possibilités, on revient de nouveau sur nos pas autant de fois que nécessaire jusqu'à avoir compéter la totalité de la grille.

4.4 Interface graphique

Pour rendre la résolution plus simple d'utilisation, nous avons décidé d'ajouter une interface graphique. Le cahier des charges qu'elle devait vérifier était assez stricte. Elle devait pouvoir afficher, avant toute chose, une grille de sudoku qui devait dès lors être facilement modifiable. Pour cela, nous avons donc créé différents menus ; l'un permettant donc l'édition de la grille, un autre permettant de choisir la méthode de résolution, ainsi qu'un menu de résolution.

Édition Lors de l'édition de la grille, un carré rouge apparaît, déplaçable avec les flèches directionnelles, il suffit alors pour modifier la case de rentrer un chiffre de 0 à 9, 0 correspondant à une case vierge. Il est également possible de sauvegarder une grille ou encore de récupérer une grille déjà enregistrée.

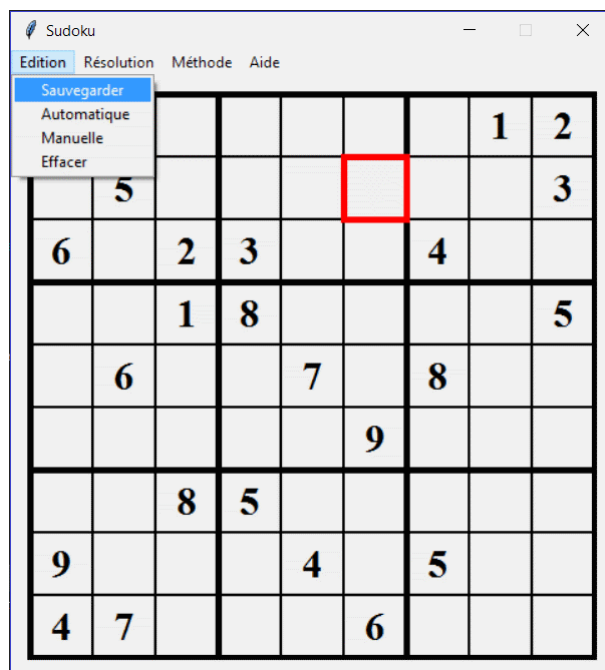


FIGURE 4.7 – Interface graphique affichant le menu Edition

Résolution Ce menu permet de lancer la résolution, et permet également de choisir entre une résolution ou une résolution pas-à-pas permettant à l'utilisateur d'observer le fonctionnement de ces algorithmes.



FIGURE 4.8 – Interface graphique affichant le menu Résolution

Méthode Le choix de la méthode se fait avec le menu *Méthode*, qui permet de choisir la méthode de résolution parmi l'*inclusion*, l'*exclusion* et le

*backtracking*¹ ou de choisir une méthode dite *globale*, s'appuyant sur tous les algorithmes, permettant ainsi d'être la plus rapide possible.

4.5 Contrôle des moteurs et écriture

4.5.1 Moteurs pas-à-pas

4.5.2 Coordonnées polaires

Par soucis de compacité de nouveau, nous avons décidé de développer une structure se basant sur les coordonnées cylindriques, permettant des dimensions maximum de 40 par 10 cm au lieu de 40 par 40 en coordonnées cartésiennes.

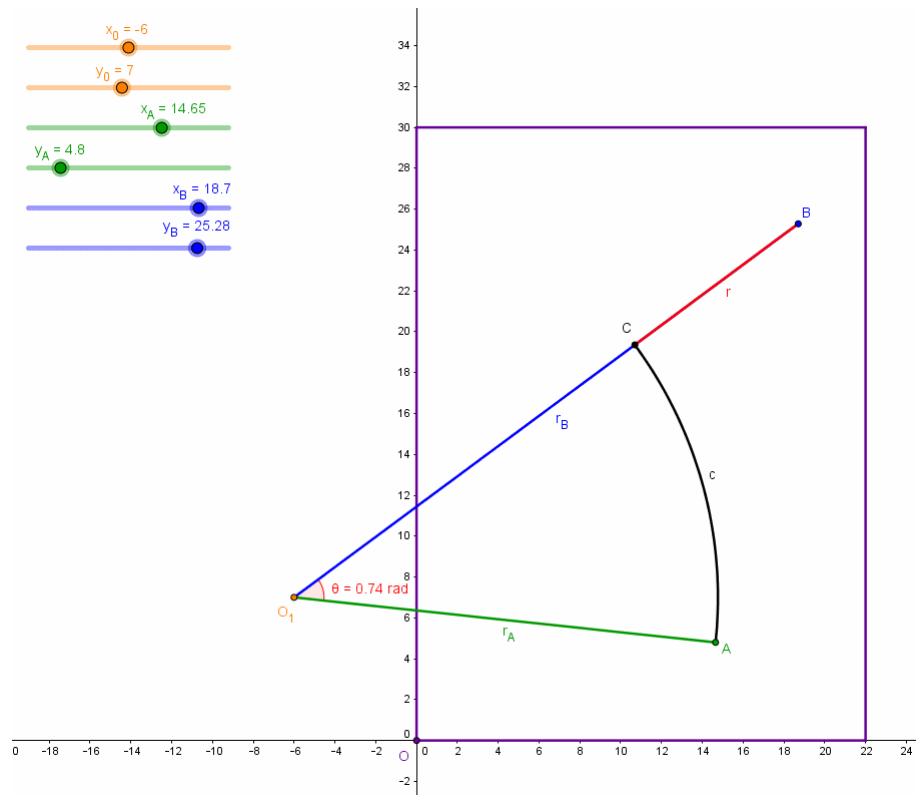


FIGURE 4.9 – Changement de repères réalisé avec *Geogebra*

4.5.3 Écriture des digits

1. Cf 4.3 Méthode de Résolution

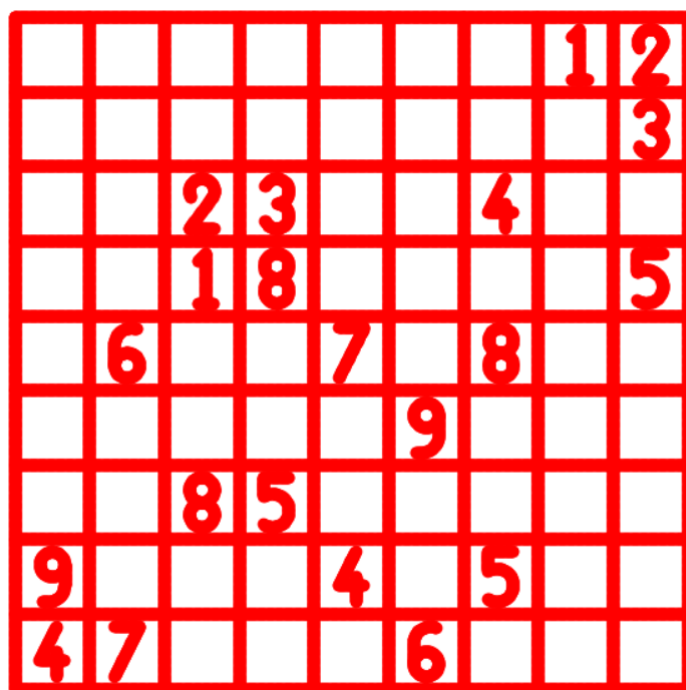


FIGURE 4.10 – Grille de sudoku en point par point tracée avec *Matplotlib*

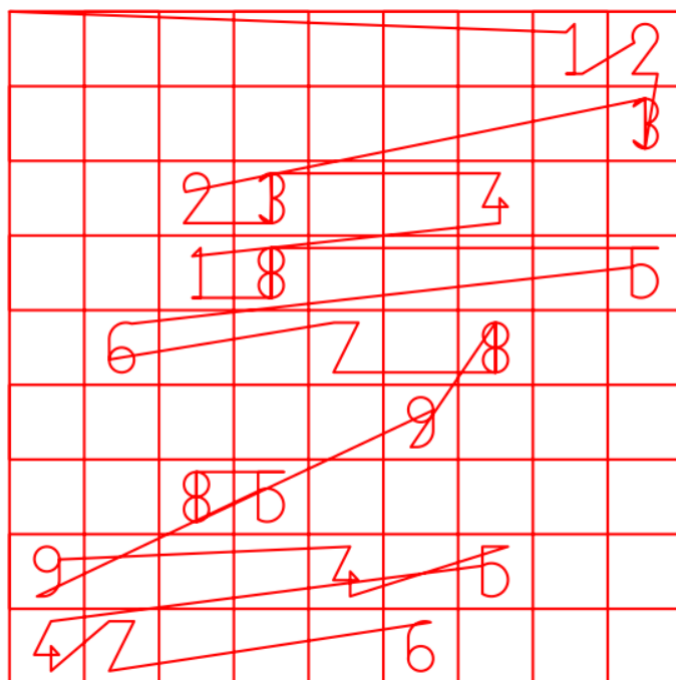


FIGURE 4.11 – Grille de sudoku tracée avec *Matplotlib*

Conclusion

Remerciements

Nous remercions M. Couvez pour ses précieux conseils et Tristan Vajente pour les impressions de pièces en 3D.

Annexe A

Fichier principal

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import os
5  import time
6  import numpy as np
7
8  import save
9  import write as w
10 import camera as cm
11 import display as dp
12 import resolution as rs
13
14
15 class Main:
16     """
17     Permet la gestion des sudoku à savoir :
18     - leur édition par l'utilisateur pour obtenir le sudoku à résoudre
19     - leur résolution à l'aide de différentes méthodes de résolution:
20         - inclusion
21         - exclusion
22         - backtracking
23         - ...
24     - leur affichage à l'aide du module tkinter
25     """
26
27     def __init__(self):
28         self.beta_version = True
29         self.error = []
30         self.taille = (3, 3)
31         self.nb_cases = self.taille[0] * self.taille[1]
32         self.sudoku = np.zeros((self.nb_cases, self.nb_cases), int)
33         self.methode_resolution = "Globale"
34         self.liste_position = []
35
36         self.W = w.Write()
37         self.Camera = cm.Camera(self)
38         self.Display = dp.Display(self)
39         self.Resolution = rs.Resolution(self)
40
41         self.Display.mainloop()
42
43
44     def setError(self, error, off=True):
45         if off:
46             if error not in self.error:
47                 self.error.append(error)
48         else:
49             if error in self.error:
50                 self.error.remove(error)
51
52     def getError(self):
53         return self.error
54
```

```

55     def setMethodeResolution(self, methode):
56         self.methode_resolution = methode
57
58     def startResolution(self, sudoku):
59         self.sudoku, self.liste_position = self.Resolution.start(sudoku, self.methode_resolution)
60         self.Display.updateSudoku(self.sudoku, self.liste_position)
61
62     def stopResolution(self):
63         pass
64
65     def setSudoku(self, sudoku):
66         self.sudoku = sudoku
67
68     def writeSudoku(self, sudoku):
69         self.sudoku = sudoku
70         save.saveSudoku(sudoku)
71         os.system("sudo python3 writing_main.py")
72
73
74 Main()

```

Annexe B

Script de résolution des sudokus

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import time
5  import numpy as np
6  from copy import copy
7
8
9  class Resolution:
10     """
11     Classe permettant de résoudre un sudoku grâce à différentes méthodes, à savoir :
12     - inclusion
13     - exclusion
14     - backtracking
15     - ...
16     Si le sudoku n'est pas résoluble, lève une erreur
17     """
18     def __init__(self, boss):
19         self.boss = boss
20         self.taille = self.boss.taille
21         self.nb_cases = self.boss.nb_cases
22         self.sudoku = self.boss.sudoku
23         self.methode_resolution = None
24         self.possibilities = []
25         self.starting_possibilities = []
26         self.resolution = False
27         self.carre = []
28         self.ligne = []
29         self.colonne = []
30
31     def beforeStart(self):
32         self.possibilities = []
33         self.carre = []
34         self.ligne = []
35         self.colonne = []
36         for i in range(self.nb_cases):
37             values = [i + 1 for i in range(self.nb_cases)]
38             self.carre.append(copy(values))
39             self.ligne.append(copy(values))
40             self.colonne.append(copy(values))
41             x = 3 * (i // 3)
42             y = 3 * (i % 3)
43             for j in range(self.nb_cases):
44                 if self.sudoku[x + j // 3, y + j % 3] != 0 \
45                     and self.sudoku[x + j // 3, y + j % 3] in self.carre[i]:
46                     self.carre[i].remove(self.sudoku[x + j // 3, y + j % 3])
47                 if self.sudoku[i, j] != 0 and self.sudoku[i, j] in self.ligne[i]:
48                     self.ligne[i].remove(self.sudoku[i, j])
49                 if self.sudoku[j, i] != 0 and self.sudoku[j, i] in self.colonne[i]:
50                     self.colonne[i].remove(self.sudoku[j, i])
51                 if not self.sudoku[i][j] and (i, j) not in self.possibilities:
52                     self.possibilities.append((i, j))
53         self.starting_possibilities = copy(self.possibilities)
54
```

```

55 def start(self, sudoku, methode):
56     self.sudoku = copy(sudoku)
57     self.beforeStart()
58     self.methode_resolution = methode
59     zero_time = time.time()
60     n = 0
61     print(self.methode_resolution)
62     if self.methode_resolution == "Backtracking":
63         self.backTracking()
64     else:
65         self.resolution = True
66         while self.resolution:
67             n += 1
68             if self.methode_resolution == "Inclusion":
69                 self.inclusion()
70             if self.methode_resolution == "Exclusion":
71                 self.exclusion()
72             if self.methode_resolution == "Globale":
73                 self.inclusion()
74                 self.exclusion()
75             if np.all(self.sudoku == sudoku):
76                 self.resolution = False
77             sudoku = copy(self.sudoku)
78             if np.any(self.sudoku == np.zeros((self.nb_cases, self.nb_cases), int)):
79                 self.methode_resolution = "Backtracking"
80                 print("Backtracking")
81                 self.backTracking()
82
83     print(time.time() - zero_time, n)
84     return self.sudoku, self.starting_possibilities
85
86 def checkListe(self, x, y):
87     """
88     Renvoie la liste des valeurs possibles pour la case de coordonnées x et y
89     :param x: int: ligne
90     :param y: int: colonne
91     :return: liste: list
92     """
93     liste = []
94     if self.sudoku[x][y] == 0:
95         liste = [i + 1 for i in range(self.nb_cases)]
96         block_x = x - x % self.taille[0]
97         block_y = y - y % self.taille[1]
98         for i in range(self.nb_cases):
99             if self.sudoku[x, i] in liste:
100                 liste.remove(self.sudoku[x, i])
101             if self.sudoku[i, y] in liste:
102                 liste.remove(self.sudoku[i, y])
103             if self.sudoku[block_x + i % self.taille[1], block_y + i // self.taille[0]] in liste:
104                 liste.remove(self.sudoku[block_x + i % self.taille[1], block_y + i // self.taille[0]])
105     return liste
106
107 def inclusion(self):
108     for x in range(self.nb_cases):
109         for y in range(self.nb_cases):
110             self.checkValues(x, y)
111
112 def exclusion(self):
113     for n in range(self.nb_cases):
114         for k in self.carre[n]:
115             x, y = 3 * (n // 3), 3 * (n % 3)
116             x_possible = []
117             y_possible = []
118             for i in range(self.taille[0]):
119                 if k in self.ligne[x + i]: x_possible.append(x + i)
120                 if k in self.colonne[y + i]: y_possible.append(y + i)
121             case_possible = []
122             for x in x_possible:
123                 for y in y_possible:
124                     if (x, y) in self.possibilities: case_possible.append((x, y))
125             self.setValuesEsclusion(case_possible, k)
126
127     j = n

```

```

128         for k in self.colonne[j]:
129             carre_possible = []
130             case_possible = []
131             for i in range(self.taille[0]):
132                 if k in self.carre[3 * i + j // 3]:
133                     carre_possible.append(3 * i + j // 3)
134             for i in range(self.nb_cases):
135                 if (i, j) in self.possibilities:
136                     if k in self.ligne[i] and (i, j) not in case_possible and \
137                         3 * (i // 3) + j // 3 in carre_possible:
138                         case_possible.append((i, j))
139             self.setValuesEsclusion(case_possible, k)
140
141         i = n
142         for k in self.ligne[i]:
143             carre_possible = []
144             case_possible = []
145             for j in range(self.taille[0]):
146                 if k in self.carre[3 * (i // 3) + j]:
147                     carre_possible.append(3 * (i // 3) + j)
148             for j in range(self.nb_cases):
149                 if (i, j) in self.possibilities:
150                     if k in self.colonne[j] and (i, j) not in case_possible and \
151                         3 * (i // 3) + j // 3 in carre_possible:
152                         case_possible.append((i, j))
153             self.setValuesEsclusion(case_possible, k)
154
155     def backTracking(self):
156         """
157         Résoud un sudoku selon la méthode de backtracking
158         :return: None or -1
159         """
160         liste_sudoku = []
161         i = 0
162         while i < len(self.possibilities):
163             x, y = self.possibilities[i]
164             liste = self.checkListe(x, y)
165             if liste:
166                 self.sudoku[x][y] = liste.pop(0)
167                 liste_sudoku.append(liste)
168                 i += 1
169             else:
170                 while not liste:
171                     i -= 1
172                     x, y = self.possibilities[i]
173                     try:
174                         liste = liste_sudoku[i]
175                     except IndexError:
176                         self.sudoku = self.boss.sudoku
177                         self.boss.setError("sudoku_insoluble")
178                         return -1
179                     if liste:
180                         self.sudoku[x][y] = liste.pop(0)
181                         liste_sudoku[i] = liste
182                         i += 1
183                         break
184                     else:
185                         liste_sudoku.pop(i)
186                         self.sudoku[x][y] = 0
187
188     def checkValues(self, x, y):
189         if (x, y) in self.possibilities:
190             possibilities = []
191             n = 3 * (x // 3) + y // 3
192             for i in range(1, self.nb_cases + 1):
193                 if i in self.ligne[x] and i in self.colonne[y] and i in self.carre[n]:
194                     possibilities.append(i)
195             self.setValues(possibilities, x, y)
196
197     def setValuesEsclusion(self, case_possible, k):
198         if len(case_possible) == 1:
199             x, y = case_possible[0][0], case_possible[0][1]
200             n = 3 * (x // 3) + y // 3

```

```

201         self.sudoku[x, y] = k
202         self.possibilities.remove((x, y))
203         self.ligne[x].remove(k)
204         self.colonne[y].remove(k)
205         self.carre[n].remove(k)
206
207     def setValues(self, possibilities, x, y):
208         if len(possibilities) == 1:
209             k = possibilities[0]
210             n = 3 * (x // 3) + y // 3
211             self.sudoku[x, y] = k
212             if (x, y) in self.possibilities: self.possibilities.remove((x, y))
213             self.ligne[x].remove(k)
214             self.colonne[y].remove(k)
215             self.carre[n].remove(k)
216
217
218 if __name__ == "__main__":
219     class Boss:
220         def __init__(self):
221             self.taille = (3, 3)
222             self.nb_cases = 9
223             self.sudoku = np.array([[3, 0, 0, 2, 0, 9, 0, 0, 5],
224                                     [0, 0, 0, 0, 0, 0, 0, 0, 0],
225                                     [0, 7, 8, 0, 0, 0, 2, 4, 0],
226                                     [0, 5, 0, 4, 0, 7, 0, 9, 0],
227                                     [0, 6, 0, 0, 2, 0, 0, 8, 0],
228                                     [0, 9, 0, 5, 0, 3, 0, 1, 0],
229                                     [0, 8, 1, 0, 0, 0, 6, 3, 0],
230                                     [0, 0, 0, 0, 0, 0, 0, 0, 0],
231                                     [7, 0, 0, 8, 0, 5, 0, 0, 1]])
232             self.Resolution = Resolution(self)
233             self.sudoku, position = self.Resolution.start(self.sudoku, "Globale")
234             print(self.sudoku)
235
236
237     Boss()

```

Annexe C

Script de gestion de la caméra

```
1  #!/usr/bin/env python3
2
3
4  class Camera:
5      """
6      Permet la gestion de la camera de la raspberry pi
7      Si celle-ci n'est pas disponible ou le module 'picamera'
8      n'a pas été installé correctement, lève une exception.
9      """
10
11     def __init__(self, boss):
12         self.boss = boss
13         self.camera = None
14         self.tryError()
15
16     def tryError(self):
17         try:
18             import picamera
19             self.camera = picamera.PiCamera()
20         except:
21             self.boss.setError("camera_error")
22
23     def takePhoto(self):
24         try:
25             self.camera.capture("Images/photos.jpg")
26             print("The photo has been taken")
27         except:
28             self.boss.setError("camera_error")
29
30
31 if __name__ == '__main__':
32     class Boss:
33         def setError(self, error):
34             if error == "module_camera":
35                 print("Le module 'picamera' n'a pas été installé correctement !")
36             if error == "disponibilite_camera":
37                 print("La caméra n'est pas disponible !")
38
39     Camera = Camera(Boss())
40     Camera.takePhoto()
```


Annexe D

Script d'affichage du sudoku

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import save
5  import numpy as np
6  from tkinter import *
7  from tkinter.messagebox import *
8
9
10 class Display(Tk):
11     """
12     Hérite de la classe Tk() qui permet l'affichage d'une fenêtre sous tkinter
13     Permet d'exploiter une interface graphique pour afficher et éditer une
14     grille de sudoku de manière plus interactive avec l'utilisateur.
15     """
16
17     def __init__(self, boss):
18         Tk.__init__(self)
19
20         self.boss = boss
21         self.color = 'black'
22         self.title("Sudoku")
23         self.resizable(width=False, height=False)
24         self.edition = False
25         self.rectangle = None
26         self.x, self.y = 0, 0
27         self.liste_position = []
28         self.taille = self.boss.taille
29         self.nb_cases = self.boss.nb_cases
30         self.sudoku = np.zeros((self.nb_cases, self.nb_cases), int)
31         self.affichage_sudoku = [[0 for i in range(self.nb_cases)] for i in range(self.nb_cases)]
32
33         # Creation des variables tkinter et widgets
34         self.Can = Canvas(width=455, height=455)
35         self.BarreMenu = self.BarreMenu(self)
36
37         # Placement des widgets
38         self.Can.grid(padx=10, pady=10)
39         self.configure(menu=self.BarreMenu)
40
41         self.createMatrix()
42         self.bind_all('<Key>', self.tryToEdit)
43
44         self.update()
45         self.setInTheMiddle()
46
47         if not self.boss.beta_version:
48             # Affiche les erreurs survenues au démarrage
49             for error in self.boss.getError():
50                 self.showError(error)
51
52     def choixMethode(self, methode):
53         self.boss.setMethodeResolution(methode)
54
```

```

55     def choixVitesse(self, vitesse):
56         self.boss.setVitesse(vitesse)
57
58     def startResolution(self):
59         self.startManualEdition(False)
60         self.boss.setError("sudoku_insoluble", False)
61         self.boss.startResolution(self.sudoku)
62
63     def beforeUsingCamera(self):
64         if "module_camera" in self.boss.getError():
65             self.showError("module_camera")
66         elif "disponibilite_camera" in self.boss.getError():
67             self.showError("disponibilite_camera")
68         else:
69             self.boss.Camera.takePhoto()
70
71     def backUp(self):
72         save.saveSudoku(self.sudoku)
73         showinfo("Sudoku", "La grille a été enregistrée avec succès !")
74
75     def openSudoku(self):
76         self.sudoku = save.readSudoku()
77         self.boss.setSudoku(self.sudoku)
78         self.updateSudoku()
79
80     def startManualEdition(self, edition=None):
81         if edition is not None:
82             self.edition = edition
83         else:
84             self.edition = not self.edition
85         if self.edition:
86             self.Can.itemconfigure(self.rectangle, width=5)
87             self.x, self.y = 0, 0
88         else:
89             self.Can.itemconfigure(self.rectangle, width=0)
90             self.update()
91
92     def createMatrix(self):
93         """
94         Permet d'afficher une grille de sudoku vide ainsi que le curseur qui est à l'origine masqué
95         :return: None
96         """
97         self.Can.delete(ALL)
98         for i in range(self.nb_cases + 1):
99             column = self.Can.create_line(50 * i + 5, 5, 50 * i + 5, 455, width=2)
100             line = self.Can.create_line(3, 50 * i + 5, 457, 50 * i + 5, width=2)
101             if i % self.taille[0] == 0:
102                 self.Can.itemconfigure(column, width=5)
103             if i % self.taille[1] == 0:
104                 self.Can.itemconfigure(line, width=5)
105             if i != self.nb_cases:
106                 for j in range(self.nb_cases):
107                     self.affichage_sudoku[i][j] = self.Can.create_text(50 * j + 30, 50 * i + 30,
108                                                                           font=('Times', 24, 'bold'), text="")
109             self.rectangle = self.Can.create_rectangle(5 + 50 * self.x, 5 + 50 * self.y, 55 + 50 * self.x,
110                                                         55 + 50 * self.y, outline='red', width=0)
111
112     def tryToEdit(self, evt):
113         key = evt.keysym
114
115         if key == 'F5':
116             self.startResolution()
117
118         if key.lower() == "b":
119             self.color = "black"
120             print("black")
121
122         if key.upper() == 'C':
123             self.boss.stopResolution()
124
125         if key.lower() == "m":
126             self.boss.writeSudoku(self.sudoku)
127

```

```

128     if key.lower() == "o":
129         self.openSudoku()
130
131     if key.lower() == "r":
132         self.color = "red"
133         print("red")
134
135     if key.lower() == "s":
136         self.backUp()
137
138     if key == "v".lower():
139         try:
140             sleep = float(input("sleep = "))
141             self.boss.setSpeed(sleep)
142         except ValueError:
143             print("a doit etre un nombre")
144
145     if key.lower() == "x":
146         self.effacerSudoku()
147
148     if key == 'Return':
149         self.startManualEdition()
150
151     if key == "BackSpace":
152         for x in range(self.nb_cases):
153             for y in range(self.nb_cases):
154                 if (x, y) in self.liste_position:
155                     self.sudoku[x][y] = 0
156             self.updateSudoku(self.sudoku, self.liste_position)
157
158     if self.edition:
159         if key == 'Right':
160             self.y += 1
161             if self.y == self.nb_cases:
162                 self.y = 0
163
164         if key == 'Left':
165             self.y -= 1
166             if self.y == -1:
167                 self.y = self.nb_cases - 1
168
169         if key == 'Down':
170             self.x += 1
171             if self.x == self.nb_cases:
172                 self.x = 0
173
174         if key == 'Up':
175             self.x -= 1
176             if self.x == -1:
177                 self.x = self.nb_cases - 1
178
179         try:
180             if 0 < int(key) < self.nb_cases + 1:
181                 self.Can.itemconfigure(self.affichage_sudoku[self.x][self.y], text=key, fill=self.color)
182             if int(key) == 0:
183                 self.Can.itemconfigure(self.affichage_sudoku[self.x][self.y], text="")
184             self.sudoku[self.x, self.y] = int(key)
185         except ValueError:
186             pass
187
188         self.Can.coords(self.rectangle, 5 + 50 * self.y, 5 + 50 * self.x, 55 + 50 * self.y, 55 + 50 * self.x)
189
190     def updateSudoku(self, sudoku=None, liste_position=[]):
191         if sudoku is None: sudoku = self.boss.sudoku
192         self.liste_position = liste_position
193         self.startManualEdition(False)
194         if "sudoku_insoluble" in self.boss.getError():
195             self.showError("sudoku_insoluble")
196         else:
197             for x in range(self.nb_cases):
198                 for y in range(self.nb_cases):
199                     if (x, y) in self.liste_position:
200                         self.Can.itemconfigure(self.affichage_sudoku[x][y], text=sudoku[x][y], fill='red')

```

```

201         else:
202             self.Can.itemconfigure(self.affichage_sudoku[x][y], text=sudoku[x][y], fill='black')
203         if sudoku[x][y] == 0:
204             self.Can.itemconfigure(self.affichage_sudoku[x][y], text="", fill='black')
205         self.sudoku = np.copy(sudoku)
206
207     def effacerSudoku(self):
208         for x in range(self.nb_cases):
209             for y in range(self.nb_cases):
210                 self.Can.itemconfigure(self.affichage_sudoku[x][y], text="", fill='black')
211         self.sudoku = np.zeros((self.nb_cases, self.nb_cases), int)
212
213     def showError(self, error):
214         if error == "module_camera":
215             showerror("Caméra", "Désolé, le module picamera n'a pas été installé correctement !")
216         elif error == "disponibilite_camera":
217             showerror("Caméra", "Désolé, la caméra n'est pas disponible !")
218         elif error == "camera_error":
219             showerror("Caméra", "Une erreur inattendue avec la camera est survenue !\n"
220                     "Veuillez réessayer ou redémarrer votre raspberry pi")
221         elif error == "sudoku_insoluble":
222             showwarning("Sudoku", "Le sudoku n'est pas résoluble")
223         else:
224             showerror("Erreur", "Une erreur inattendue est survenue !")
225
226     def showAide(self):
227         showinfo("Aide", "Le menu 'Aide' n'est pas encore disponible !")
228
229     def showAPropos(self):
230         showinfo("Aide", "Le menu 'A Propos' n'est pas encore disponible !")
231
232     def setInTheMiddle(self):
233         geo = []
234         s = ''
235         for i in self.geometry():
236             if i.isdigit():
237                 s += i
238             else:
239                 geo.append(s)
240                 s = ''
241         geo.append(s)
242         new_geo = geo[0] + "x" + geo[1] + "+" + str(int(0.5 * (self.winfo_screenwidth() - int(geo[0])))) \
243             + "+" + str(int(0.5 * (self.winfo_screenheight() - int(geo[1]))))
244         self.geometry(new_geo)
245
246     class BarreMenu(Menu):
247         """
248         Hérite de la classe Menu() qui permet à l'utilisateur de définir
249         ses préférences et les opérations qui doivent être effectuées
250         """
251
252     def __init__(self, boss):
253         Menu.__init__(self)
254         self.boss = boss
255         self.mode_resolution = IntVar()
256         self.vitesse = IntVar()
257
258         # Sélection par défaut des items
259         self.mode_resolution.set(0)
260         self.vitesse.set(0)
261
262         # Création des menus
263         self.menu_edition = Menu(self, tearoff=0)
264         self.menu_resolution = Menu(self, tearoff=0)
265         self.menu_methode = Menu(self, tearoff=0)
266         self.menu_aide = Menu(self, tearoff=0)
267
268         # Ajout des menus
269         self.add_cascade(label="Edition", menu=self.menu_edition)
270         self.add_cascade(label="Résolution", menu=self.menu_resolution)
271         self.add_cascade(label="Méthode", menu=self.menu_methode)
272         self.add_cascade(label="Aide", menu=self.menu_aide)
273

```

```

274     # Ajout des items du menu 'Edition'
275     self.menu_edition.add_command(label="Sauvegarder", command=self.boss.backUp)
276     self.menu_edition.add_command(label="Automatique", command=self.boss.openSudoku)
277     self.menu_edition.add_command(label="Manuelle", command=self.boss.startManualEdition)
278     self.menu_edition.add_command(label="Effacer", command=self.boss.effacerSudoku)
279
280     # Ajout des items du menu 'Résolution'
281     self.menu_resolution.add_radiobutton(label="Directe", value=0, variable=self.vitesse,
282                                         command=lambda: self.boss.choixVitesse("Directe"))
283     self.menu_resolution.add_radiobutton(label="Pas à pas", value=1, variable=self.vitesse,
284                                         command=lambda: self.boss.choixVitesse("Pas à pas"))
285
286     self.menu_resolution.add_separator()
287     self.menu_resolution.add_command(label="Lancer", command=self.boss.startResolution)
288
289     # Ajout des items du menu "Méthode"
290     self.menu_methode.add_radiobutton(label="Globale", value=0, variable=self.mode_resolution,
291                                       command=lambda: self.boss.choixMethode("Globale"))
292     self.menu_methode.add_radiobutton(label="Inclusion", value=1, variable=self.mode_resolution,
293                                       command=lambda: self.boss.choixMethode("Inclusion"))
294     self.menu_methode.add_radiobutton(label="Exclusion", value=2, variable=self.mode_resolution,
295                                       command=lambda: self.boss.choixMethode("Exclusion"))
296     self.menu_methode.add_radiobutton(label="Backtracking", value=3, variable=self.mode_resolution,
297                                       command=lambda: self.boss.choixMethode("Backtracking"))
298
299     # Ajout des items du menu "Aide"
300     self.menu_aide.add_command(label="Fonctionnement", command=self.boss.showAide)
301     self.menu_aide.add_command(label="A Propos", command=self.boss.showAPropos)

```

Annexe E

Script de gestion des moteurs pas-à-pas

```
1  #!/usr/bin/env/python3
2  # -*- coding: utf-8 -*-
3
4  import time
5  import math
6  import threading
7  try:
8      error = None
9      import RPi.GPIO as GPIO
10 except ImportError:
11     error = "module_GPIO"
12
13
14 class InitMoveMotor:
15     """
16     Vérifie que le module RPi.GPIO permettant de gérer les sorties/entrées GPIO
17     de la raspberry a été importé correctement,
18     définit les sorties GPIO de la Raspberry utilisées pour contrôler les moteurs,
19     initialise les processus des moteurs (motor1 et motor2),
20     déplace les moteurs simultanément pour se rendre d'un point à un autre
21     """
22
23     def __init__(self):
24         self.tryError()
25         self.beta_version = True
26         self.bobines_motor1 = (29, 31, 33, 35)
27         self.bobines_motor2 = (7, 11, 13, 15)
28         self.turn_on_led = 19
29         self.working_led = 21
30         self.M = Point(7.5, 7)
31         self.points = [(22, 7), (16.2, 24.07), (4.3, 14.93)]
32         self.r_step = 0.0203
33         self.theta_step = 0.0056
34
35         self.pinInit()
36
37         self.motor1 = Motor(self.bobines_motor1)
38         self.motor2 = Motor(self.bobines_motor2)
39         self.turnOnLed = BlinkingLed(self.turn_on_led)
40         self.workingLed = BlinkingLed(self.working_led, True)
41
42         self.motor1.start()
43         self.motor2.start()
44         self.turnOnLed.start()
45         self.workingLed.start()
46
47         self.initializePosition()
48         if self.points: self.movingMotor()
49
50     def initializePosition(self):
51         """
52         Déplace le moteur de sorte de le placer en position initiale
53         :return: None
54         """
```

```

55     pass
56
57 def movingMotor(self):
58     self.sleep(False)
59     n = max(1, len(self.points) // 1000)
60     while self.points:
61         x_b, y_b = self.points[0]
62         B = Point(x_b, y_b)
63         r = B.r - self.M.r
64         theta = B.theta - self.M.theta
65         print(r, theta)
66         nb_steps1 = int(r / self.r_step + 0.5)
67         nb_steps2 = int(theta / self.theta_step + 0.5)
68         print(nb_steps1, nb_steps2)
69         if abs(nb_steps1) > abs(nb_steps2):
70             self.motor1.speed, self.motor2.speed = self.setTime(nb_steps1, nb_steps2)
71         else:
72             self.motor2.speed, self.motor1.speed = self.setTime(nb_steps2, nb_steps1)
73         print(self.motor1.speed, self.motor2.speed)
74         self.motor1.nb_steps = nb_steps1
75         self.motor2.nb_steps = nb_steps2
76         while self.motor1.nb_steps != 0 or self.motor2.nb_steps != 0:
77             time.sleep(0.1)
78             self.M = Point(x_b, y_b)
79             self.points.pop(0)
80         self.sleep()
81
82 def setTime(self, nb_step_a, nb_step_b):
83     speed_a = 10
84     if nb_step_b != 0:
85         speed_b = abs(nb_step_a * speed_a / nb_step_b)
86     else:
87         speed_b = 0
88     return speed_a, speed_b
89
90 def startMoving(self):
91     power = True
92
93 def pinInit(self):
94     """
95     Initialise les pins de la raspberry pour contrôler le moteur
96     :return: None
97     """
98     if not error:
99         GPIO.setmode(GPIO.BOARD)
100         GPIO.setwarnings(False)
101         for i in range(4):
102             GPIO.setup(self.bobines_motor1[i], GPIO.OUT)
103             GPIO.setup(self.bobines_motor2[i], GPIO.OUT)
104     else: print(error)
105
106 def tryError(self):
107     """
108     Renvoie une erreur le cas échéant
109     :return:
110     """
111     pass
112
113 def setPoints(self, points):
114     self.points = points
115
116 def getPoints(self):
117     return self.points
118
119 def sleep(self, sleep=True):
120     if sleep:
121         self.motor1.sleep()
122         self.motor2.sleep()
123         self.workingLed.sleep()
124         self.turnOnLed.sleep(False)
125     else:
126         self.workingLed.sleep(False)
127         self.turnOnLed.sleep()

```

```

128
129     def stop(self):
130         self.motor1.stop()
131         self.motor2.stop()
132         self.workingLed.stop()
133         self.turnOnLed.stop()
134         if not error: GPIO.cleanup()
135
136
137 class Motor(threading.Thread):
138     """
139     Permet de piloter les moteurs pas-à-pas indépendamment l'un de l'autre
140     """
141     nb = 1
142     def __init__(self, bobines=0, position=0, nb_steps=0, speed=100):
143         threading.Thread.__init__(self)
144         self.number = Motor.nb
145         self.bobines = bobines
146         self.position = position
147         self.nb_steps = nb_steps
148         self.speed = speed
149         self.power = True
150         self.motor_alim = [[1, 0, 0, 1], [0, 1, 0, 1], [0, 1, 1, 0], [1, 0, 1, 0]]
151         Motor.nb += 1
152
153     def run(self):
154         while self.power:
155             if self.nb_steps != 0:
156                 time.sleep(self.speed / 1000)
157                 self.moveMotor()
158             else:
159                 time.sleep(0.1)
160
161     def moveMotor(self):
162         if self.nb_steps != 0:
163             rotation = int(self.nb_steps / abs(self.nb_steps))
164             self.nb_steps -= rotation
165             self.position += rotation
166             self.setPins()
167
168     def setPins(self):
169         if not error:
170             for i in range(4):
171                 GPIO.output(self.bobines[i], self.motor_alim[self.position % 4][i])
172
173     def sleep(self):
174         print("motor{}: sleep".format(self.number))
175         if not error:
176             for i in range(4):
177                 GPIO.output(self.bobines[i], 0)
178
179     def stop(self):
180         print("motor{}: stop".format(self.number))
181         self.power = False
182
183     def setPower(self, power):
184         self.power = power
185
186
187 class ManualMotor(Motor):
188     def __init__(self, bobines, position=0, nb_steps=0, speed=100):
189         Motor.__init__(self)
190         self.bobines = bobines
191         self.position = position
192         self.nb_steps = nb_steps
193         self.speed = speed
194         self.move = False
195         self.direction = 1
196
197     def run(self):
198         while self.power:
199             if self.move:
200                 time.sleep(self.speed / 1000)

```



```

201         self.moveMotor()
202     else:
203         time.sleep(0.1)
204
205     def moveMotor(self):
206         self.position += self.direction
207
208
209 class BlinkingLed(threading.Thread):
210     def __init__(self, led_pin, sleeping=False):
211         threading.Thread.__init__(self)
212         self.power = True
213         self.sleeping = sleeping
214         self.led_pin = led_pin
215         self.pinInit()
216
217     def run(self):
218         while self.power:
219             if self.sleeping or error:
220                 time.sleep(2)
221             else:
222                 GPIO.output(self.led_pin, 1)
223                 time.sleep(0.1)
224                 GPIO.output(self.led_pin, 0)
225                 time.sleep(1)
226
227     def pinInit(self):
228         if not error:
229             GPIO.setmode(GPIO.BOARD)
230             GPIO.setup(self.led_pin, GPIO.OUT)
231
232     def stop(self):
233         self.power = False
234
235     def sleep(self, sleeping=True):
236         self.sleeping = sleeping
237
238
239 class Origin:
240     def __init__(self):
241         self.x_0 = -6
242         self.y_0 = 7
243
244     def coords(self):
245         return self.x_0, self.y_0
246
247
248 class Point(Origin):
249     def __init__(self, x, y, coordinate="cartesian"):
250         Origin.__init__(self)
251         if coordinate == "cartesian":
252             self.x, self.y = x, y
253             self.newPolarCoords()
254         else:
255             self.r = x
256             self.theta = y
257             self.newCartesianCoords()
258
259     def newPolarCoords(self):
260         self.r = math.sqrt((self.x - self.x_0) ** 2 + (self.y - self.y_0) ** 2)
261         if self.y - self.y_0 < 0:
262             self.theta = -math.acos((self.x - self.x_0) / self.r)
263         else:
264             self.theta = math.acos((self.x - self.x_0) / self.r)
265
266     def newCartesianCoords(self):
267         self.x = self.r * math.cos(self.theta) + self.x_0
268         self.y = self.r * math.sin(self.theta) + self.y_0
269
270
271 if __name__ == "__main__":
272     init = InitMoveMotor()
273     time.sleep(5)

```

274 `init.stop()`

Annexe F

Script permettant l'écriture d'un sudoku

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import math
5  import numpy as np
6  import matplotlib.pyplot as plt
7
8
9  class Write:
10     def __init__(self):
11         self.x, self.y = [], []
12         self.step = 0.17
13         self.coordinate = [(5, 20), (20, 5)]
14         self.a, self.b = self.coordinate[0][0], self.coordinate[0][1]
15         self.c, self.d = self.coordinate[1][0], self.coordinate[1][1]
16         self.nx = (self.c - self.a) / 9
17         self.ny = (self.b - self.d) / 9
18         self.x0 = self.a + self.nx / 2
19         self.y0 = self.d + self.ny / 2
20         self.L = 2 / 3 * min(self.nx, self.ny)
21
22     def writeOne(self, x0, y0):
23         x = - self.L / 12
24         while x <= self.L / 12:
25             y = x + 5 * self.L / 12
26             self.x.append(x + x0)
27             self.y.append(y + y0)
28             x += self.step
29         x = self.L / 12
30         y = self.L / 2
31         while y >= - self.L / 2:
32             self.x.append(x + x0)
33             self.y.append(y + y0)
34             y -= self.step
35         x = - self.L / 12
36         y = - self.L / 2
37         while x <= self.L / 4:
38             self.x.append(x + x0)
39             self.y.append(y + y0)
40             x += self.step
41
42     def writeTwo(self, x0, y0):
43         liste_x, liste_y = [], []
44         y = self.L / 8
45         x = -10
46         while x < -self.L / 4:
47             try:
48                 x = - math.sqrt((self.L / 4) ** 2 - (y - self.L / 4) ** 2)
49             except ValueError:
50                 x = 0
51             liste_x.append(x)
52             liste_y.append(y)
53             y += self.step
54         liste_x.append(- self.L / 4)
```

```

55     liste_y.append(self.L / 4)
56     while x <= 0:
57         y = self.L / 4 + math.sqrt((self.L / 4) ** 2 - x ** 2)
58         liste_x.append(x)
59         liste_y.append(y)
60         x += self.step
61     liste_x.append(0)
62     liste_y.append(self.L / 2)
63     l = len(liste_x)
64     for i in range(l - 1, -1, -1):
65         liste_x.append(- liste_x[i])
66         liste_y.append(liste_y[i])
67     x, y = liste_x[-1], liste_y[-1]
68     a = (5 * self.L / 8) / (x + self.L / 4)
69     b = self.L / 2 * (a / 2 - 1)
70     x -= self.step
71     while x > - self.L / 4:
72         y = a * x + b
73         liste_x.append(x)
74         liste_y.append(y)
75         x -= self.step
76     x = - self.L / 4
77     y = - self.L / 2
78     while x <= self.L / 4:
79         liste_x.append(x)
80         liste_y.append(y)
81         x += self.step
82     for i in range(len(liste_x)):
83         self.x.append(liste_x[i] + x0)
84         self.y.append(liste_y[i] + y0)
85
86     def writeThree(self, x0, y0):
87         liste_x, liste_y = [], []
88         y = 0
89         x = 0
90         while x < self.L / 6:
91             y = self.L / 4 + math.sqrt((self.L / 4) ** 2 - x ** 2)
92             liste_x.append(x)
93             liste_y.append(y)
94             x += self.step
95         while y >= self.L / 4:
96             x = math.sqrt((self.L / 4) ** 2 - (y - self.L / 4) ** 2)
97             liste_x.append(x)
98             liste_y.append(y)
99             y -= self.step
100        l = len(liste_x)
101        if self.L / 4 not in liste_x or self.L / 4 not in liste_y:
102            liste_x.append(self.L / 4)
103            liste_y.append(self.L / 4)
104        liste_x.append(0)
105        liste_y.append(0)
106        for i in range(l, -1, -1):
107            if liste_x[i] > 0:
108                liste_x.append(liste_x[i])
109                liste_y.append(self.L / 2 - liste_y[i])
110        l = len(liste_x)
111        liste_x.append(0)
112        liste_y.append(self.L / 2)
113        for i in range(l - 1, -1, -1):
114            if liste_y[i] > self.L / 3:
115                liste_x.append(- liste_x[i])
116                liste_y.append(liste_y[i])
117        for i in range(len(liste_x)):
118            liste_x.append(liste_x[i])
119            liste_y.append(-liste_y[i])
120        l = len(liste_x)
121        for i in range(l):
122            self.x.append(liste_x[l - 1 - i] + x0)
123            self.y.append(liste_y[l - 1 - i] + y0)
124
125    def writeFour(self, x0, y0):
126        liste_x, liste_y = [], []
127        x, y = self.L / 12, self.L / 2

```

```

128     while x <= self.L / 4:
129         liste_x.append(x)
130         liste_y.append(y)
131         if y > - self.L / 6:
132             y -= self.step
133             x = y / 2 - self.L / 6
134         else:
135             x += self.step
136     x, y = self.L / 12, 0
137     while y >= - self.L / 2:
138         liste_x.append(x)
139         liste_y.append(y)
140         y -= self.step
141     for i in range(len(liste_x)):
142         self.x.append(liste_x[i] + x0)
143         self.y.append(liste_y[i] + y0)
144
145 def writeFive(self, x0, y0):
146     x, y = self.L / 4, self.L / 2
147     y1 = - self.L / 6 + self.L * math.sqrt(1 / 12)
148     while y > y1:
149         self.x.append(x + x0)
150         self.y.append(y + y0)
151         if x > - self.L / 4:
152             x -= self.step
153         else:
154             y -= self.step
155     liste_x, liste_y = [], []
156     while x < self.L / 6:
157         try:
158             y = - self.L / 6 + math.sqrt(self.L ** 2 / 9 - (x + self.L / 12) ** 2)
159         except ValueError: y = 0
160         liste_x.append(x)
161         liste_y.append(y)
162         x += self.step
163     while y > - self.L / 6:
164         x = - self.L / 12 + math.sqrt(self.L ** 2 / 9 - (y + self.L / 6) ** 2)
165         liste_x.append(x)
166         liste_y.append(y)
167         y -= self.step
168     l = len(liste_x)
169     for i in range(l - 1, -1, -1):
170         liste_x.append(liste_x[i])
171         liste_y.append(- self.L / 3 - liste_y[i])
172     liste_x.append(self.L / 4)
173     liste_y.append(-self.L / 6)
174     l = len(liste_x)
175     for i in range(l):
176         self.x.append(liste_x[l - 1 - i] + x0)
177         self.y.append(liste_y[l - 1 - i] + y0)
178
179 def writeSix(self, x0, y0):
180     x, y = self.L / 5, self.L / 2
181     while x > - self.L / 6:
182         y = self.L / 6 + math.sqrt(self.L ** 2 / 9 - (x - self.L / 12) ** 2)
183         self.x.append(x + x0)
184         self.y.append(y + y0)
185         x -= self.step
186     while y > - self.L / 4:
187         if y > self.L / 6:
188             x = self.L / 12 - math.sqrt(self.L ** 2 / 9 - (y - self.L / 6) ** 2)
189             self.x.append(x + x0)
190             self.y.append(y + y0)
191             y -= self.step
192     liste_x, liste_y = [], []
193     while x < - self.L / 6:
194         try:
195             x = - math.sqrt(self.L ** 2 / 16 - (y + self.L / 4) ** 2)
196         except ValueError:
197             x = 0
198             y = 0
199         liste_x.append(x)
200         liste_y.append(y)

```

```

201     y -= self.step
202 while x <= 0:
203     y = - self.L / 4 - math.sqrt(self.L ** 2 / 16 - x ** 2)
204     liste_x.append(x)
205     liste_y.append(y)
206     x += self.step
207 l = len(liste_x)
208 for i in range(l - 1, -1, -1):
209     liste_x.append(- liste_x[i])
210     liste_y.append(liste_y[i])
211 l = len(liste_x)
212 for i in range(l - 1, -1, -1):
213     liste_x.append(liste_x[i])
214     liste_y.append(- self.L / 2 - liste_y[i])
215 l = len(liste_x)
216 for i in range(l):
217     self.x.append(liste_x[i] + x0)
218     self.y.append(liste_y[i] + y0)
219
220 def writeSeven(self, x0, y0):
221     x = - self.L / 4
222     y = self.L / 2
223     while x <= self.L / 4:
224         self.x.append(x + x0)
225         self.y.append(y + y0)
226         x += self.step
227     x = self.L / 4
228     while y >= - self.L / 2:
229         x = y / 2
230         self.x.append(x + x0)
231         self.y.append(y + y0)
232         y -= self.step
233     self.x.append(-self.L / 4 + x0)
234     self.y.append(-self.L / 2 + y0)
235
236 def writeEight(self, x0, y0):
237     liste_x, liste_y = [], []
238     y = 0
239     x = 0
240     while x < self.L / 6:
241         y = self.L / 4 + math.sqrt((self.L / 4) ** 2 - x ** 2)
242         liste_x.append(x)
243         liste_y.append(y)
244         x += self.step
245     while y >= self.L / 4:
246         x = math.sqrt((self.L / 4) ** 2 - (y - self.L / 4) ** 2)
247         liste_x.append(x)
248         liste_y.append(y)
249         y -= self.step
250     l = len(liste_x)
251     if self.L / 4 not in liste_x or self.L / 4 not in liste_y:
252         liste_x.append(self.L / 4)
253         liste_y.append(self.L / 4)
254     for i in range(l, -1, -1):
255         liste_x.append(liste_x[i])
256         liste_y.append(self.L / 2 - liste_y[i])
257     l = len(liste_x)
258     liste_x.append(0)
259     liste_y.append(self.L / 2)
260     for i in range(l - 1, -1, -1):
261         liste_x.append(- liste_x[i])
262         liste_y.append(liste_y[i])
263     for i in range(len(liste_x)):
264         liste_x.append(liste_x[i])
265         liste_y.append(-liste_y[i])
266     l = len(liste_x)
267     for i in range(l):
268         self.x.append(liste_x[l - 1 - i] + x0)
269         self.y.append(liste_y[l - 1 - i] + y0)
270
271 def writeNine(self, x0, y0):
272     x, y = - self.L / 5, - self.L / 2
273     while x < - self.L / 12:

```

```

274         self.x.append(x + x0)
275         self.y.append(y + y0)
276         x += self.step
277     while x < self.L / 6:
278         y = - self.L / 6 - math.sqrt(self.L ** 2 / 9 - (x + self.L / 12) ** 2)
279         self.x.append(x + x0)
280         self.y.append(y + y0)
281         x += self.step
282     while y < self.L / 4:
283         if y < - self.L / 6:
284             try:
285                 x = - self.L / 12 + math.sqrt(self.L ** 2 / 9 - (y + self.L / 6) ** 2)
286             except ValueError: x = 0
287             self.x.append(x + x0)
288             self.y.append(y + y0)
289             y += self.step
290     liste_x, liste_y = [], []
291     while x > self.L / 6:
292         try:
293             x = math.sqrt(self.L ** 2 / 16 - (y - self.L / 4) ** 2)
294         except ValueError:
295             x = 0
296             y = 0
297         liste_x.append(x)
298         liste_y.append(y)
299         y += self.step
300     while x >= 0:
301         y = self.L / 4 + math.sqrt(self.L ** 2 / 16 - x ** 2)
302         liste_x.append(x)
303         liste_y.append(y)
304         x -= self.step
305     l = len(liste_x)
306     for i in range(l - 1, -1, -1):
307         liste_x.append(- liste_x[i])
308         liste_y.append(liste_y[i])
309     l = len(liste_x)
310     for i in range(l - 1, -1, -1):
311         liste_x.append(liste_x[i])
312         liste_y.append(self.L / 2 - liste_y[i])
313     l = len(liste_x)
314     for i in range(l):
315         self.x.append(liste_x[i] + x0)
316         self.y.append(liste_y[i] + y0)
317
318     def writeNumbers(self, n, x0, y0):
319         if n == 1: self.writeOne(x0, y0)
320         if n == 2: self.writeTwo(x0, y0)
321         if n == 3: self.writeThree(x0, y0)
322         if n == 4: self.writeFour(x0, y0)
323         if n == 5: self.writeFive(x0, y0)
324         if n == 6: self.writeSix(x0, y0)
325         if n == 7: self.writeSeven(x0, y0)
326         if n == 8: self.writeEight(x0, y0)
327         if n == 9: self.writeNine(x0, y0)
328
329     def writeLine(self, x0, y0, x1, y1):
330         x = x0
331         if x1 == x0:
332             y = y0
333             if y0 < y1:
334                 while y < y1:
335                     self.x.append(x)
336                     self.y.append(y)
337                     y += self.step
338             else:
339                 while y > y1:
340                     self.x.append(x)
341                     self.y.append(y)
342                     y -= self.step
343         else:
344             a = (y1 - y0) / (x1 - x0)
345             b = y0 - a * x0
346             if x0 < x1:

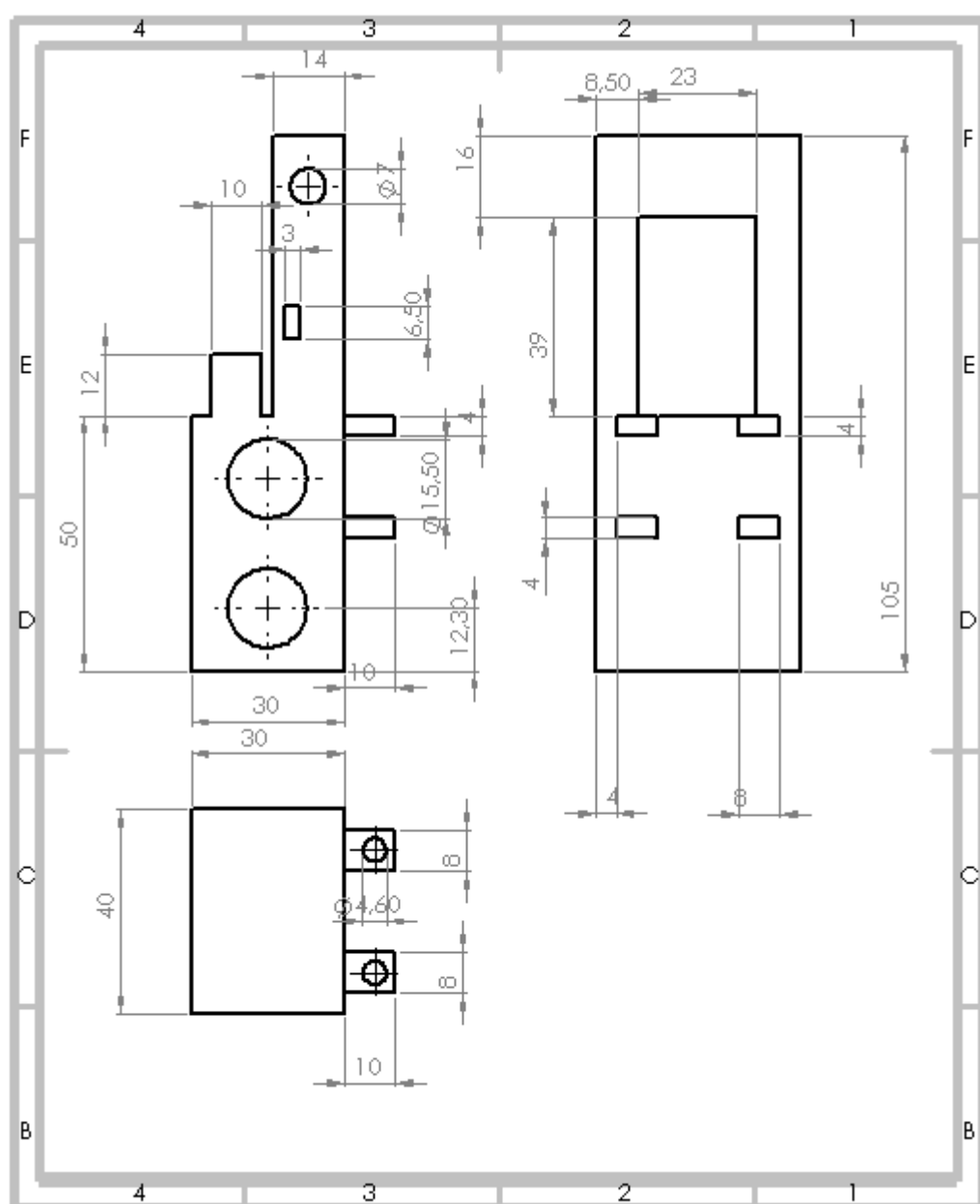
```

```

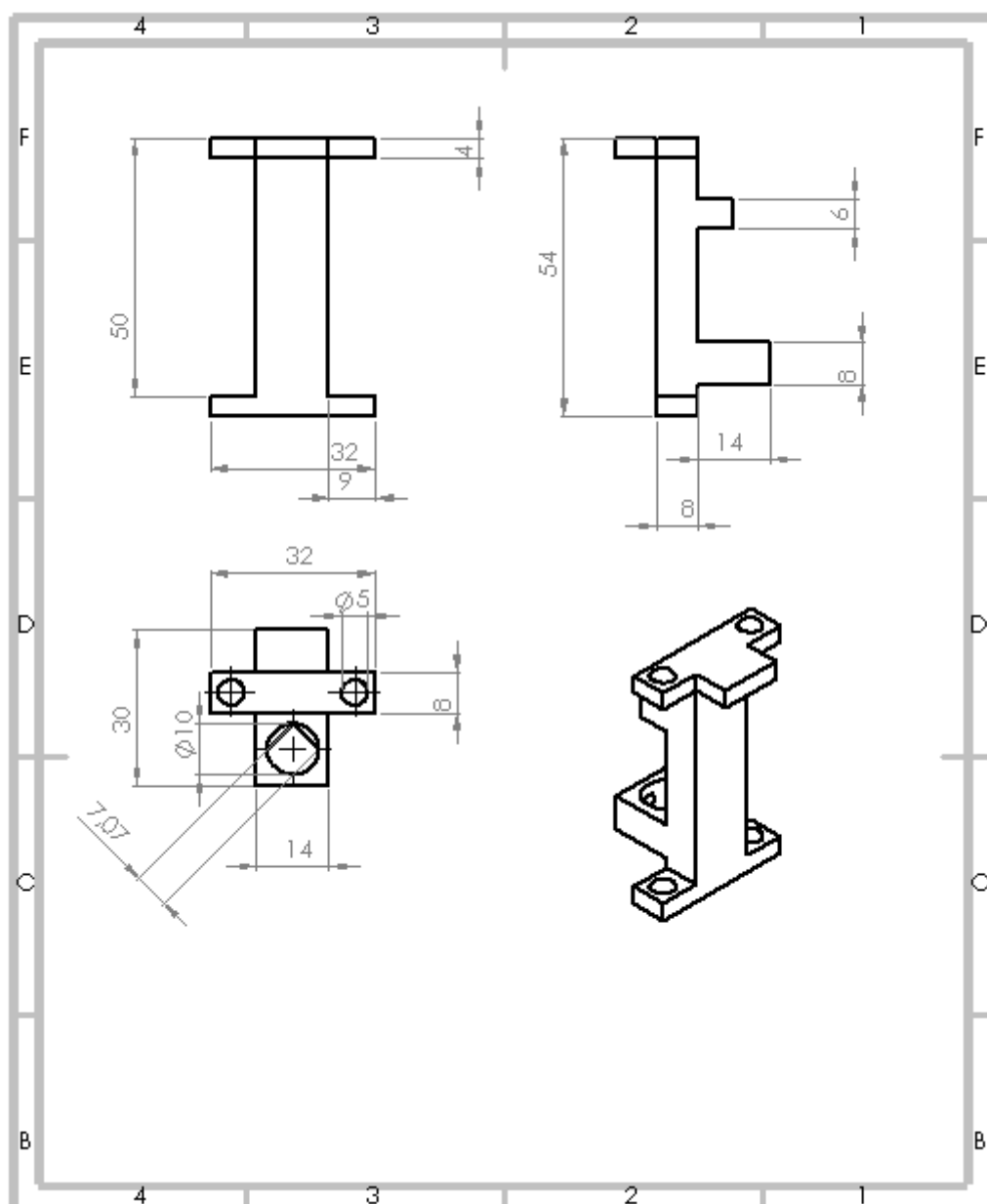
347         while x < x1:
348             y = a * x + b
349             self.x.append(x)
350             self.y.append(y)
351             x += self.step
352         else:
353             while x > x1:
354                 y = a * x + b
355                 self.x.append(x)
356                 self.y.append(y)
357                 x -= self.step
358
359     def writeSudoku(self, sudoku):
360         for i in range(10):
361             if i % 2 == 0:
362                 self.writeLine(self.a, self.d + self.ny * i, self.c, self.d + self.ny * i)
363             else:
364                 self.writeLine(self.c, self.d + self.ny * i, self.a, self.d + self.ny * i)
365         for i in range(10):
366             if i % 2 == 0:
367                 self.writeLine(self.a + self.nx * (9 - i), self.b, self.a + self.nx * (9 - i), self.d)
368             else:
369                 self.writeLine(self.a + self.nx * (9 - i), self.d, self.a + self.nx * (9 - i), self.b)
370         for i in range(9):
371             for j in range(9):
372                 self.writeNumbers(sudoku[i][j], self.x0 + self.nx * j,
373                                   self.y0 + self.ny * (8 - i))
374         points = []
375         for i in range(len(self.x)):
376             points.append((self.x[i], self.y[i]))
377         return points
378
379     def write(self):
380         plt.plot(self.x, self.y, 'r')
381         plt.grid(True)
382         plt.axis('equal')
383         plt.axis('off')
384         plt.show()
385
386
387 if __name__ == "__main__":
388     w = Write()
389     sudoku = np.array([[0, 0, 0, 0, 0, 0, 0, 1, 2],
390                        [0, 0, 0, 0, 0, 0, 0, 0, 3],
391                        [0, 0, 2, 3, 0, 0, 4, 0, 0],
392                        [0, 0, 1, 8, 0, 0, 0, 0, 5],
393                        [0, 6, 0, 0, 7, 0, 8, 0, 0],
394                        [0, 0, 0, 0, 0, 9, 0, 0, 0],
395                        [0, 0, 8, 5, 0, 0, 0, 0, 0],
396                        [9, 0, 0, 0, 4, 0, 5, 0, 0],
397                        [4, 7, 0, 0, 0, 0, 6, 0, 0]])
398     print(w.writeSudoku(sudoku))
399     w.write()

```

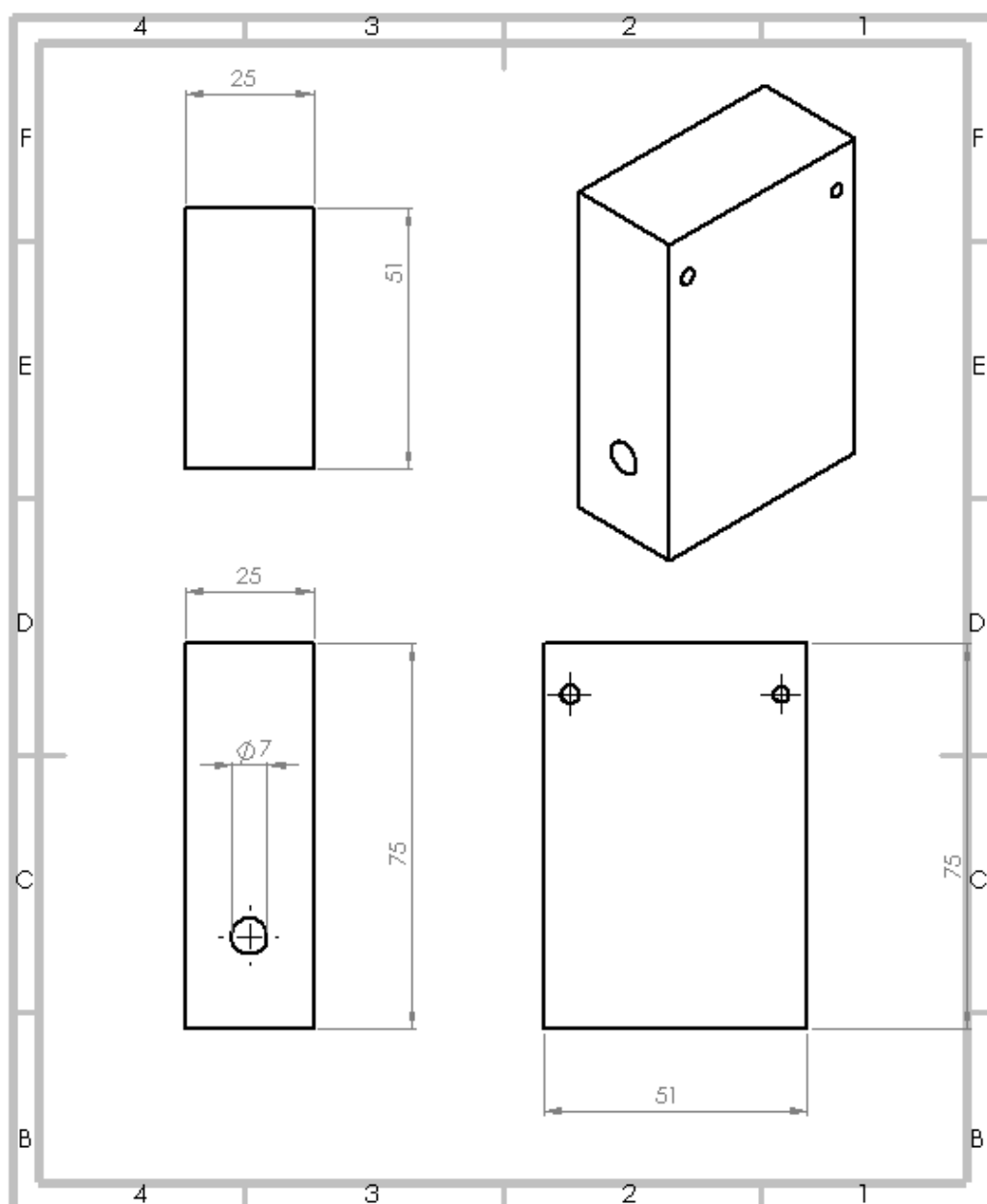

Mise en plan du chariot



Mise en plan du support du stylo



Mise en plan du lien chariot/caméra



Mise en plan du support de la caméra

