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

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 remplacé la breadboard par une

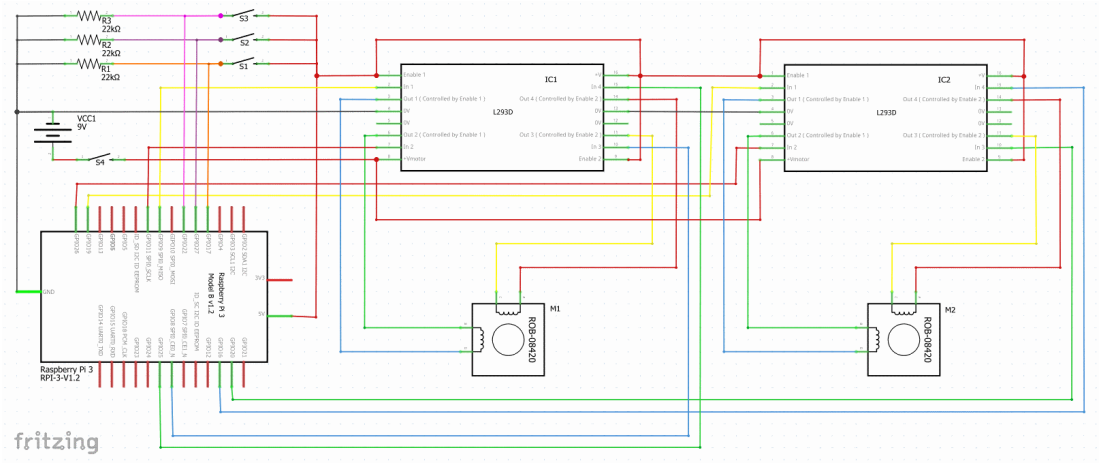


FIGURE 2.1 – Schéma électrique réalisé avec *Fritzing*

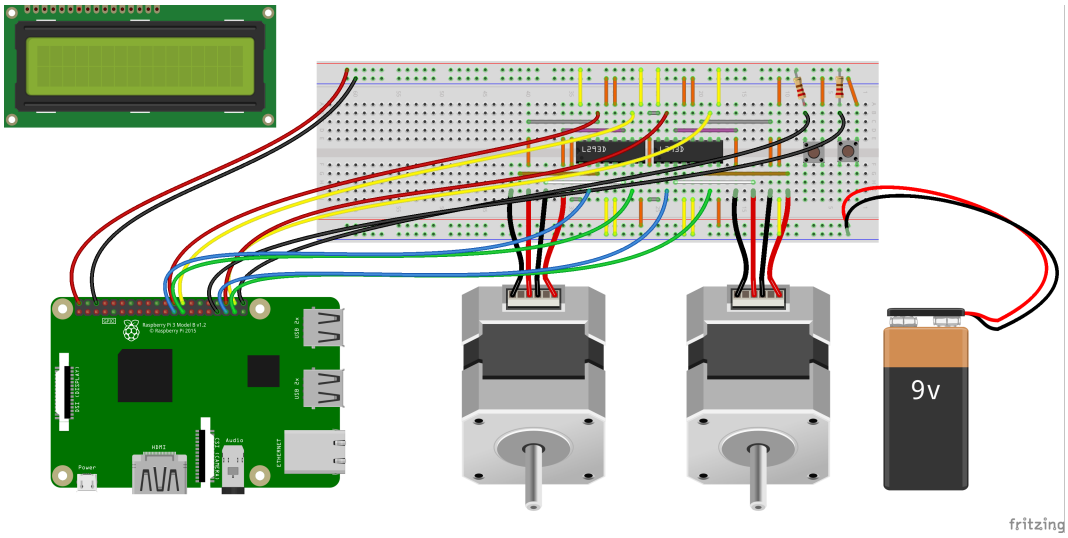


FIGURE 2.2 – Premier montage

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 avions 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.

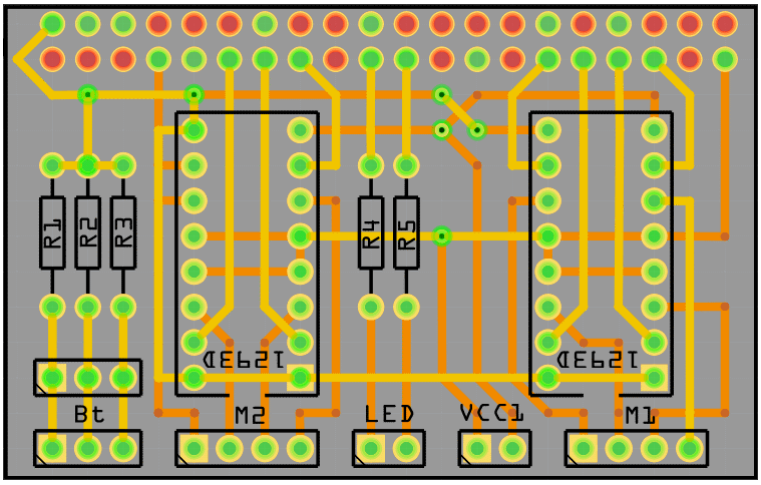


FIGURE 2.3 – Circuit imprimé réalisé avec *Fritzing*

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 Difficultés rencontrés et solutions mises en place

3.2 Conception du chariot en 3D

Chapitre 4

Informatique

4.1 Présentation globale

Tous les algorithmes développé dans le cadre de ce projet sont disponibles en annexe. Ils ont été, pour la plupart, développé en Python 3, les autres se basant sur Python 2 car certaines bibliothèques dont nous avions besoin, notamment pour la reconnaissance, n'était disponible 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 :

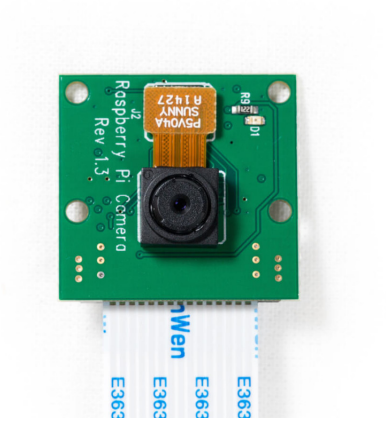


FIGURE 4.1 – Caméra Raspberry Pi V2

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

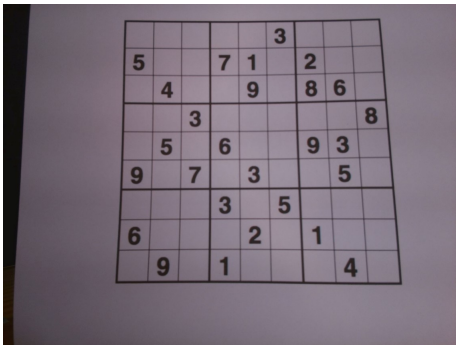


FIGURE 4.2 – Exemple de grille de sudoku

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 :

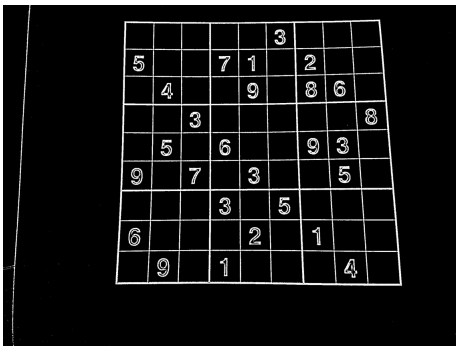


FIGURE 4.3 – Exemple de grille "seuillée"

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.

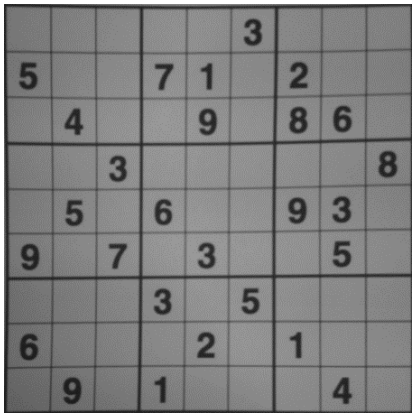


FIGURE 4.4 – Exemple de grille découpée sans perspective

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

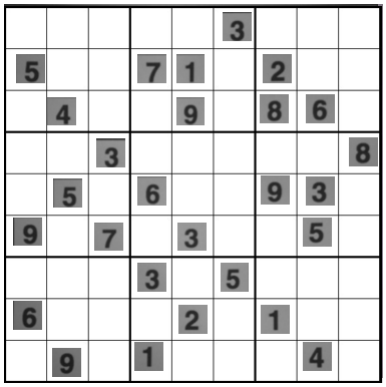


FIGURE 4.5 – Exemple de grille où chaque chiffre a été découpé

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 apparait, 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.

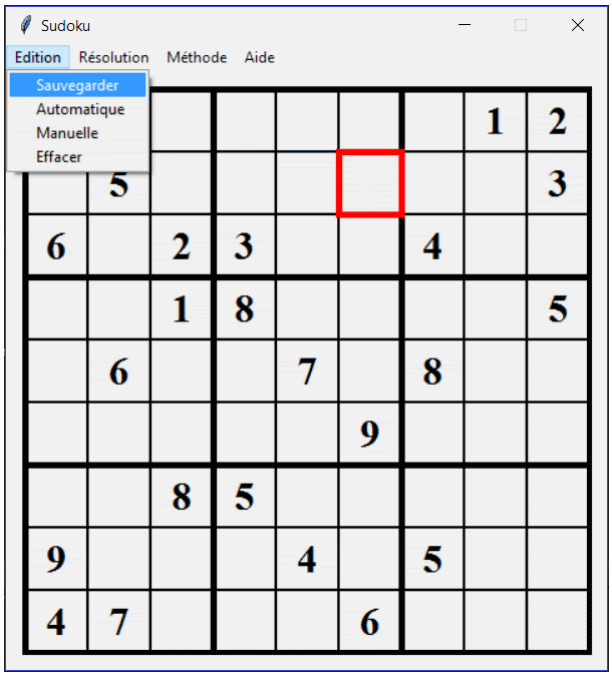


FIGURE 4.7 – Interface graphique affichant le menu Edition

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

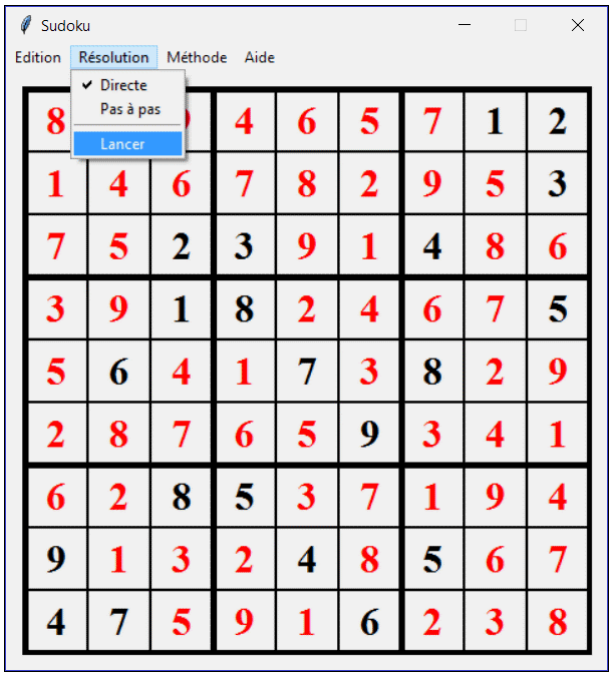


FIGURE 4.8 – Interface graphique affichant le menu Édition

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

Moteurs pas-à-pas

Coordonnées polaires

Par soucis de compacité, 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 cartésiens. En effet, en coordonnées cartésien, pour rendre le système stable, il aurait fallu placer une tige de chaque côté de la feuille, sur lesquelles se déplacerait une plateforme pouvant déplacer le stylo selon la largeur de la feuille. De plus, l'utilisation des coordonnées polaires ne sont que très peu utilisés pour ce type de système, ce qui se révélait donc plus intéressant à développer.

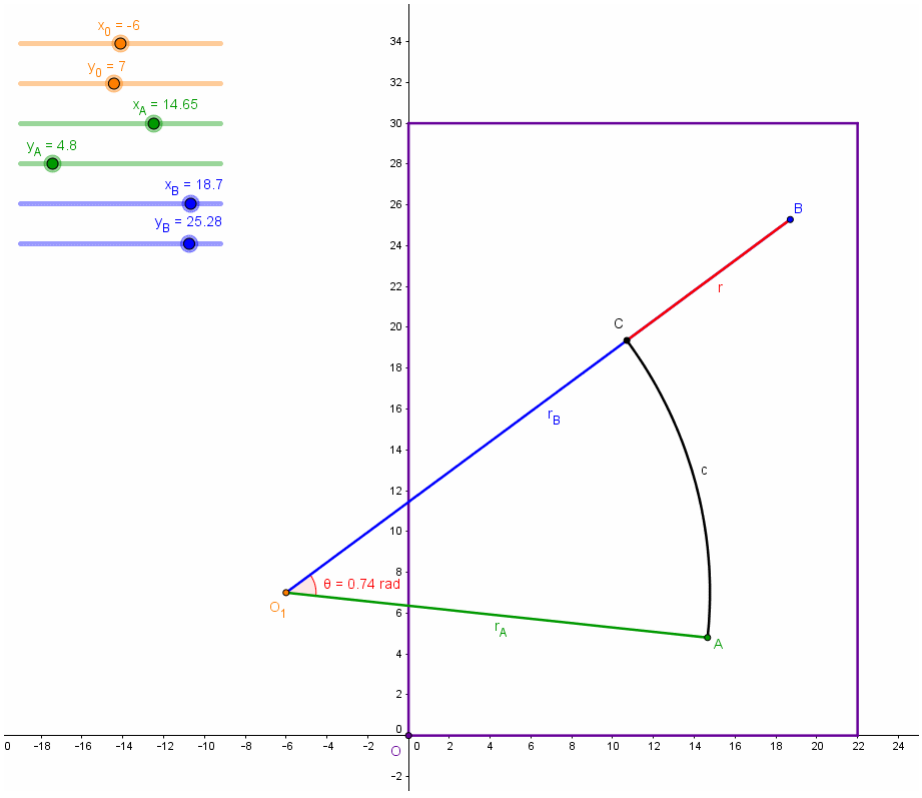


FIGURE 4.9 – Changement de repères réalisé avec *Geogebra*

Écriture des digits

Nous souhaitions dès le début pouvoir écrire des grilles de différentes tailles, et non des grilles de tailles fixes. C'est pourquoi nous avons basé le script d'écriture sur du point par point, permettant de choisir la précision théorique

1. Cf **??** Méthode de Résolution

des chiffres tracés. Cependant les moteurs pas-à-pas ne pouvant se déplacer que d'un pas fixé de 1.8 ° , une trop grande précision ne pourrait entraînerait que des déplacements nuls des moteurs, c'est pourquoi cette manière de programmation permet un compromis entre précision théorique et précision pratique.

FIGURE 4.10 – Chiffres d'une précision théorique assez faible

En augmentant la précision théorique, on obtient des chiffres quasiment continus permettant d'obtenir la figure suivante.

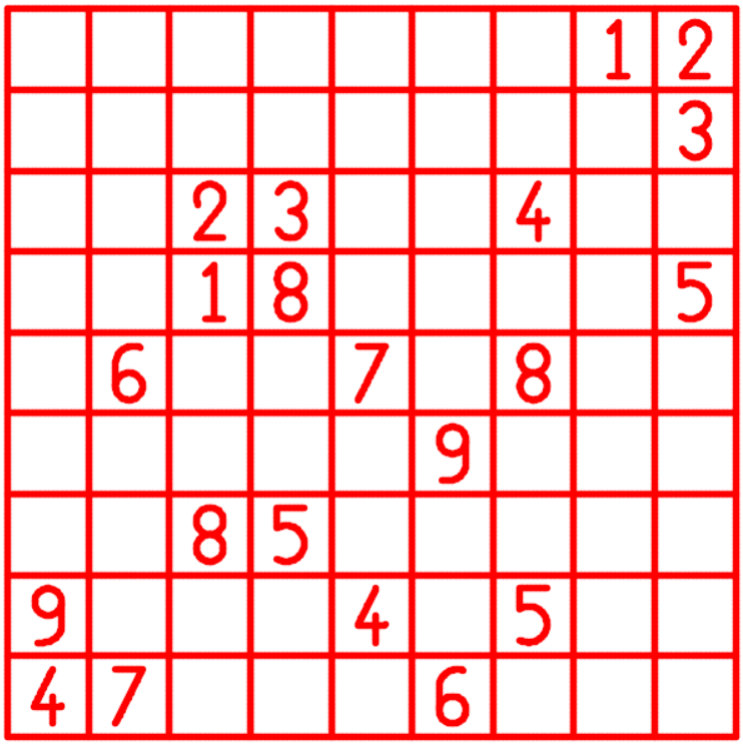


FIGURE 4.11 – Grille de sudoku en point par point tracée avec *MatplotLib*

Il faut prévoir dans le script la possibilité d'envoyé une impulsion au servomoteur afin qu'il lève le stylo lorsqu'un chiffre a été tracée afin d'éviter que les chiffres ne soient reliés entre eux, comme le suggère la figure suivante.

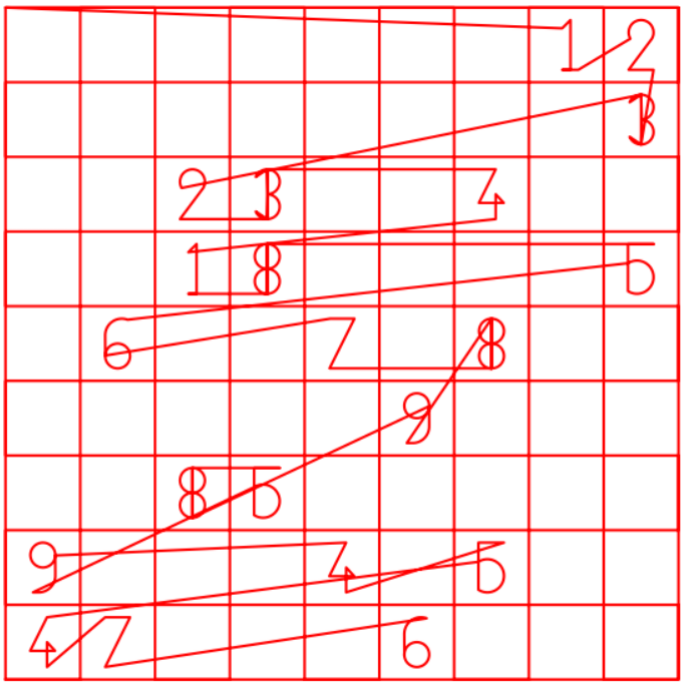


FIGURE 4.12 – Grille de sudoku tracée avec *MatplotLib*

Conclusion

Tous les scipts et ce dossier sont disponibles sur GitHub [2]

2. www.github.com/alphter/Sudoku-Plotter/

Remerciements

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

Annexe A

Fichier principal

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import os
import time
import numpy as np

import save
import write as w
import camera as cm
import display as dp
import resolution as rs


class Main:
    """
    Permet la gestion des sudoku à savoir :
        - leur édition par l'utilisateur pour obtenir le sudoku à résoudre
        - leur résolution à l'aide de différentes méthodes de résolution:
            - inclusion
            - exclusion
            - backtracking
            - ...
        - leur affichage à l'aide du module tkinter
    """

    def __init__(self):
        self.beta_version = True
        self.error = []
        self.taille = (3, 3)
        self.nb_cases = self.taille[0] * self.taille[1]
        self.sudoku = np.zeros((self.nb_cases, self.nb_cases), int)
        self.methode_resolution = "Globale"
        self.liste_position = []

        self.W = w.Write()
        self.Camera = cm.Camera(self)
        self.Display = dp.Display(self)
        self.Resolution = rs.Resolution(self)

        self.Display.mainloop()


    def setError(self, error, off=True):
        if off:
            if error not in self.error:
                self.error.append(error)
        else:
            if error in self.error:
                self.error.remove(error)

    def getError(self):
        return self.error

```

```
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

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import time
import numpy as np
from copy import copy


class Resolution:
    """
    Classe permettant de résoudre un sudoku grâce à différentes méthodes, à savoir :
        - inclusion
        - exclusion
        - bactracking
        - ...
    Si le sudoku n'est pas résoluble, lève une erreur
    """
    def __init__(self, boss):
        self.boss = boss
        self.taille = self.boss.taille
        self.nb_cases = self.boss.nb_cases
        self.sudoku = self.boss.sudoku
        self.methode_resolution = None
        self.possibilities = []
        self.starting_possibilities = []
        self.resolution = False
        self.carre = []
        self.ligne = []
        self.colonne = []

    def beforeStart(self):
        self.possibilities = []
        self.carre = []
        self.ligne = []
        self.colonne = []
        for i in range(self.nb_cases):
            values = [i + 1 for i in range(self.nb_cases)]
            self.carre.append(copy(values))
            self.ligne.append(copy(values))
            self.colonne.append(copy(values))
            x = 3 * (i // 3)
            y = 3 * (i % 3)
            for j in range(self.nb_cases):
                if self.sudoku[x + j // 3, y + j % 3] != 0 \
                        and self.sudoku[x + j // 3, y + j % 3] in self.carre[i]:
                    self.carre[i].remove(self.sudoku[x + j // 3, y + j % 3])
                if self.sudoku[i, j] != 0 and self.sudoku[i, j] in self.ligne[i]:
                    self.ligne[i].remove(self.sudoku[i, j])
                if self.sudoku[j, i] != 0 and self.sudoku[j, i] in self.colonne[i]:
                    self.colonne[i].remove(self.sudoku[j, i])
                if not self.sudoku[i][j] and (i, j) not in self.possibilities:
                    self.possibilities.append((i, j))
        self.starting_possibilities = copy(self.possibilities)

```

```
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

```python
#!/usr/bin/env python3


class Camera:
    """
    Permet la gestion de la camera de la raspberry pi
    Si celle-ci n'est pas disponible ou le module 'picamera'
    n'a pas été installé correctement, lève une exception.
    """

    def __init__(self, boss):
        self.boss = boss
        self.camera = None
        self.tryError()

    def tryError(self):
        try:
            import picamera
            self.camera = picamera.PiCamera()
        except:
            self.boss.setError("camera_error")

    def takePhoto(self):
        try:
            self.camera.capture("Images/photos.jpg")
            print("The photo has been taken")
        except:
            self.boss.setError("camera_error")


if __name__ == '__main__':
    class Boss:
        def setError(self, error):
            if error == "module_camera":
                print("Le module 'picamera' n'a pas été installé correctement !")
            if error == "disponibilite_camera":
                print("La caméra n'est pas disponible !")

    Camera = Camera(Boss())
    Camera.takePhoto()
```

Annexe D

Script d'affichage du sudoku

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import save
import numpy as np
from tkinter import *
from tkinter.messagebox import *


class Display(Tk):
    """
    Hérite de la classe Tk() qui permet l'affichage d'une fenêtre sous tkinter
    Permet d'exploiter une interface graphique pour afficher et éditer une
    grille de sudoku de manière plus interactive avec l'utilisateur.
    """

    def __init__(self, boss):
        Tk.__init__(self)

        self.boss = boss
        self.color = 'black'
        self.title("Sudoku")
        self.resizable(width=False, height=False)
        self.edition = False
        self.rectangle = None
        self.x, self.y = 0, 0
        self.liste_position = []
        self.taille = self.boss.taille
        self.nb_cases = self.boss.nb_cases
        self.sudoku = np.zeros((self.nb_cases, self.nb_cases), int)
        self.affichage_sudoku = [[0 for i in range(self.nb_cases)] for i in range(self.nb_cases)]

        # Creation des variables tkinter et widgets
        self.Can = Canvas(width=455, height=455)
        self.BarreMenu = self.BarreMenu(self)

        # Placement des widgets
        self.Can.grid(padx=10, pady=10)
        self.configure(menu=self.BarreMenu)

        self.createMatrix()
        self.bind_all('<Key>', self.tryToEdit)

        self.update()
        self.setInTheMiddle()

        if not self.boss.beta_version:
            # Affiche les erreurs survenues au démarrage
            for error in self.boss.getError():
                self.showError(error)

    def choixMethode(self, methode):
        self.boss.setMethodeResolution(methode)

```

```
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')
```

```
                    else:
                        self.Can.itemconfigure(self.affichage_sudoku[x][y], text=sudoku[x][y], fill='black')
                    if sudoku[x][y] == 0:
                        self.Can.itemconfigure(self.affichage_sudoku[x][y], text="", fill='black')
            self.sudoku = np.copy(sudoku)

    def effacerSudoku(self):
        for x in range(self.nb_cases):
            for y in range(self.nb_cases):
                self.Can.itemconfigure(self.affichage_sudoku[x][y], text="", fill='black')
        self.sudoku = np.zeros((self.nb_cases, self.nb_cases), int)

    def showError(self, error):
        if error == "module_camera":
            showerror("Caméra", "Désolé, le module picamera n'a pas été installé correctement !")
        elif error == "disponibilite_camera":
            showerror("Caméra", "Désolé, la caméra n'est pas disponible !")
        elif error == "camera_error":
            showerror("Caméra", "Une erreur inattendue avec la camera est survenue !\n"
                                "Veuillez réessayer ou redémarrer votre raspberry pi")
        elif error == "sudoku_insoluble":
            showwarning("Sudoku", "Le sudoku n'est pas résoluble")
        else:
            showerror("Erreur", "Une erreur inattendue est survenue !")

    def showAide(self):
        showinfo("Aide", "Le menu 'Aide' n'est pas encore disponible !")

    def showAPropos(self):
        showinfo("Aide", "Le menu 'A Propos' n'est pas encore disponible !")

    def setInTheMiddle(self):
        geo = []
        s = ''
        for i in self.geometry():
            if i.isdigit():
                s += i
            else:
                geo.append(s)
                s = ''
        geo.append(s)
        new_geo = geo[0] + "x" + geo[1] + "+" + str(int(0.5 * (self.winfo_screenwidth() - int(geo[0])))) \
                  + "+" + str(int(0.5 * (self.winfo_screenheight() - int(geo[1]))))
        self.geometry(new_geo)

    class BarreMenu(Menu):
        """
        Hérite de la classe Menu() qui permet à l'utilisateur de définir
        ses préférences et les opérations qui doivent être efféctuées
        """

        def __init__(self, boss):
            Menu.__init__(self)
            self.boss = boss
            self.mode_resolution = IntVar()
            self.vitesse = IntVar()

            # Sélection par défaut des items
            self.mode_resolution.set(0)
            self.vitesse.set(0)

            # Création des menus
            self.menu_edition = Menu(self, tearoff=0)
            self.menu_resolution = Menu(self, tearoff=0)
            self.menu_methode = Menu(self, tearoff=0)
            self.menu_aide = Menu(self, tearoff=0)

            # Ajout des menus
            self.add_cascade(label="Edition", menu=self.menu_edition)
            self.add_cascade(label="Résolution", menu=self.menu_resolution)
            self.add_cascade(label="Méthode", menu=self.menu_methode)
            self.add_cascade(label="Aide", menu=self.menu_aide)

```

```
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

```python
#!/usr/bin/env/python3
# -*- coding: utf-8 -*-

import time
import math
import threading
try:
    error = None
    import RPi.GPIO as GPIO
except ImportError:
    error = "module_GPIO"


class InitMoveMotor:
    """
    Vérifie que le module RPI.GPIO permettant de gérer les sorties/entrées GPIO
    de la raspberry a été importé correctement,
    définit les sorties GPIO de la Raspberry utilisées pour controler les moteurs,
    initialise les processus des moteurs (motor1 et motor2),
    déplace les moteurs simultanément pour se rendre d'un point à un autre
    """

    def __init__(self):
        self.tryError()
        self.beta_version = True
        self.bobines_motor1 = (29, 31, 33, 35)
        self.bobines_motor2 = (7, 11, 13, 15)
        self.turn_on_led = 19
        self.working_led = 21
        self.M = Point(7.5, 7)
        self.points = [(22, 7), (16.2, 24.07), (4.3, 14.93)]
        self.r_step = 0.0203
        self.theta_step = 0.0056

        self.pinInit()

        self.motor1 = Motor(self.bobines_motor1)
        self.motor2 = Motor(self.bobines_motor2)
        self.turnOnLed = BlinkingLed(self.turn_on_led)
        self.workingLed = BlinkingLed(self.working_led, True)

        self.motor1.start()
        self.motor2.start()
        self.turnOnLed.start()
        self.workingLed.start()

        self.initializePosition()
        if self.points: self.movingMotor()

    def initializePosition(self):
        """
        Déplace le moteur de sorte de le placer en position initiale
        :return: None
        """
```

```
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 errur le cas échéant
109         :return:
110         """
111         pass
112
113     def setPoins(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)
```

```
                self.moveMotor()
            else:
                time.sleep(0.1)

    def moveMotor(self):
        self.position += self.direction


class BlinkingLed(threading.Thread):
    def __init__(self, led_pin, sleeping=False):
        threading.Thread.__init__(self)
        self.power = True
        self.sleeping = sleeping
        self.led_pin = led_pin
        self.pinInit()

    def run(self):
        while self.power:
            if self.sleeping or error:
                time.sleep(2)
            else:
                GPIO.output(self.led_pin, 1)
                time.sleep(0.1)
                GPIO.output(self.led_pin, 0)
                time.sleep(1)

    def pinInit(self):
        if not error:
            GPIO.setmode(GPIO.BOARD)
            GPIO.setup(self.led_pin, GPIO.OUT)

    def stop(self):
        self.power = False

    def sleep(self, sleeping=True):
        self.sleeping = sleeping


class Origin:
    def __init__(self):
        self.x_0 = -6
        self.y_0 = 7

    def coords(self):
        return self.x_0, self.y_0


class Point(Origin):
    def __init__(self, x, y, coordinate="cartesian"):
        Origin.__init__(self)
        if coordinate == "cartesian":
            self.x, self.y = x, y
            self.newPolarCoords()
        else:
            self.r = x
            self.theta = y
            self.newCartesianCoords()

    def newPolarCoords(self):
        self.r = math.sqrt((self.x - self.x_0) ** 2 + (self.y - self.y_0) ** 2)
        if self.y - self.y_0 < 0:
            self.theta = -math.acos((self.x - self.x_0) / self.r)
        else:
            self.theta = math.acos((self.x - self.x_0) / self.r)

    def newCartesianCoords(self):
        self.x = self.r * math.cos(self.theta) + self.x_0
        self.y = self.r * math.sin(self.theta) + self.y_0


if __name__ == "__main__":
    init = InitMoveMotor()
    time.sleep(5)
```

```
274         init.stop()
```

Annexe F

Script permettant l'écriture d'un sudoku

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import math
import numpy as np
import matplotlib.pyplot as plt


class Write:
    def __init__(self):
        self.x, self.y = [], []
        self.step = 0.0017
        self.coordinate = [(5, 20), (20, 5)]
        self.a, self.b = self.coordinate[0][0], self.coordinate[0][1]
        self.c, self.d = self.coordinate[1][0], self.coordinate[1][1]
        self.nx = (self.c - self.a) / 9
        self.ny = (self.b - self.d) / 9
        self.x0 = self.a + self.nx / 2
        self.y0 = self.d + self.ny / 2
        self.L = 2 / 3 * min(self.nx, self.ny)

    def append(self, liste_x, liste_y, x0, y0):
        for i in range(len(liste_x)):
            self.x.append(liste_x[i] + x0)
            self.y.append(liste_y[i] + y0)

    def writeOne(self, x0, y0):
        x = - self.L / 12
        while x <= self.L / 12:
            y = x + 5 * self.L / 12
            self.x.append(x + x0)
            self.y.append(y + y0)
            x += self.step
        x = self.L / 12
        y = self.L / 2
        while y >= - self.L / 2:
            self.x.append(x + x0)
            self.y.append(y + y0)
            y -= self.step
        x = - self.L / 12
        y = - self.L / 2
        while x < self.L / 4:
            self.x.append(x + x0)
            self.y.append(y + y0)
            x += self.step
        self.x.append(self.L / 4 + x0)
        self.y.append(y + y0)

    def writeTwo(self, x0, y0):
        liste_x, liste_y = [], []
        y = self.L / 8
        x = -10
        while y < 3 * self.L / 7:
            try:
```

```
                x = - math.sqrt((self.L / 4) ** 2 - (y - self.L / 4) ** 2)
            except ValueError:
                x = 0
            liste_x.append(x)
            liste_y.append(y)
            y += self.step
        liste_x.append(- self.L / 4)
        liste_y.append(self.L / 4)
        while x <= 0:
            y = self.L / 4 + math.sqrt((self.L / 4) ** 2 - x ** 2)
            liste_x.append(x)
            liste_y.append(y)
            x += self.step
        liste_x.append(0)
        liste_y.append(self.L / 2)
        l = len(liste_x)
        for i in range(l - 1, -1, -1):
            liste_x.append(- liste_x[i])
            liste_y.append(liste_y[i])
        x, y = liste_x[-1], liste_y[-1]
        a = (5 * self.L / 8) / (x + self.L / 4)
        b = self.L / 2 * (a / 2 - 1)
        x -= self.step
        while x > - self.L / 4:
            y = a * x + b
            liste_x.append(x)
            liste_y.append(y)
            x -= self.step
        x = - self.L / 4
        y = - self.L / 2
        while x < self.L / 4:
            liste_x.append(x)
            liste_y.append(y)
            x += self.step
        liste_x.append(self.L / 4)
        liste_y.append(y)
        self.append(liste_x, liste_y, x0, y0)

    def writeThree(self, x0, y0):
        liste_x, liste_y = [], []
        y = 0
        x = 0
        while x < self.L / 6:
            y = self.L / 4 + math.sqrt((self.L / 4) ** 2 - x ** 2)
            liste_x.append(x)
            liste_y.append(y)
            x += self.step
        while y >= self.L / 4:
            x = math.sqrt((self.L / 4) ** 2 - (y - self.L / 4) ** 2)
            liste_x.append(x)
            liste_y.append(y)
            y -= self.step
        l = len(liste_x)
        if self.L / 4 not in liste_x or self.L / 4 not in liste_y:
            liste_x.append(self.L / 4)
            liste_y.append(self.L / 4)
        liste_x.append(0)
        liste_y.append(0)
        for i in range(l, -1, -1):
            if liste_x[i] > 0:
                liste_x.append(liste_x[i])
                liste_y.append(self.L / 2 - liste_y[i])
        l = len(liste_x)
        liste_x.append(0)
        liste_y.append(self.L / 2)
        for i in range(l - 1, -1, -1):
            if liste_y[i] > self.L / 3:
                liste_x.append(- liste_x[i])
                liste_y.append(liste_y[i])
        for i in range(len(liste_x)):
            liste_x.append(liste_x[i])
            liste_y.append(-liste_y[i])
        l = len(liste_x)
```

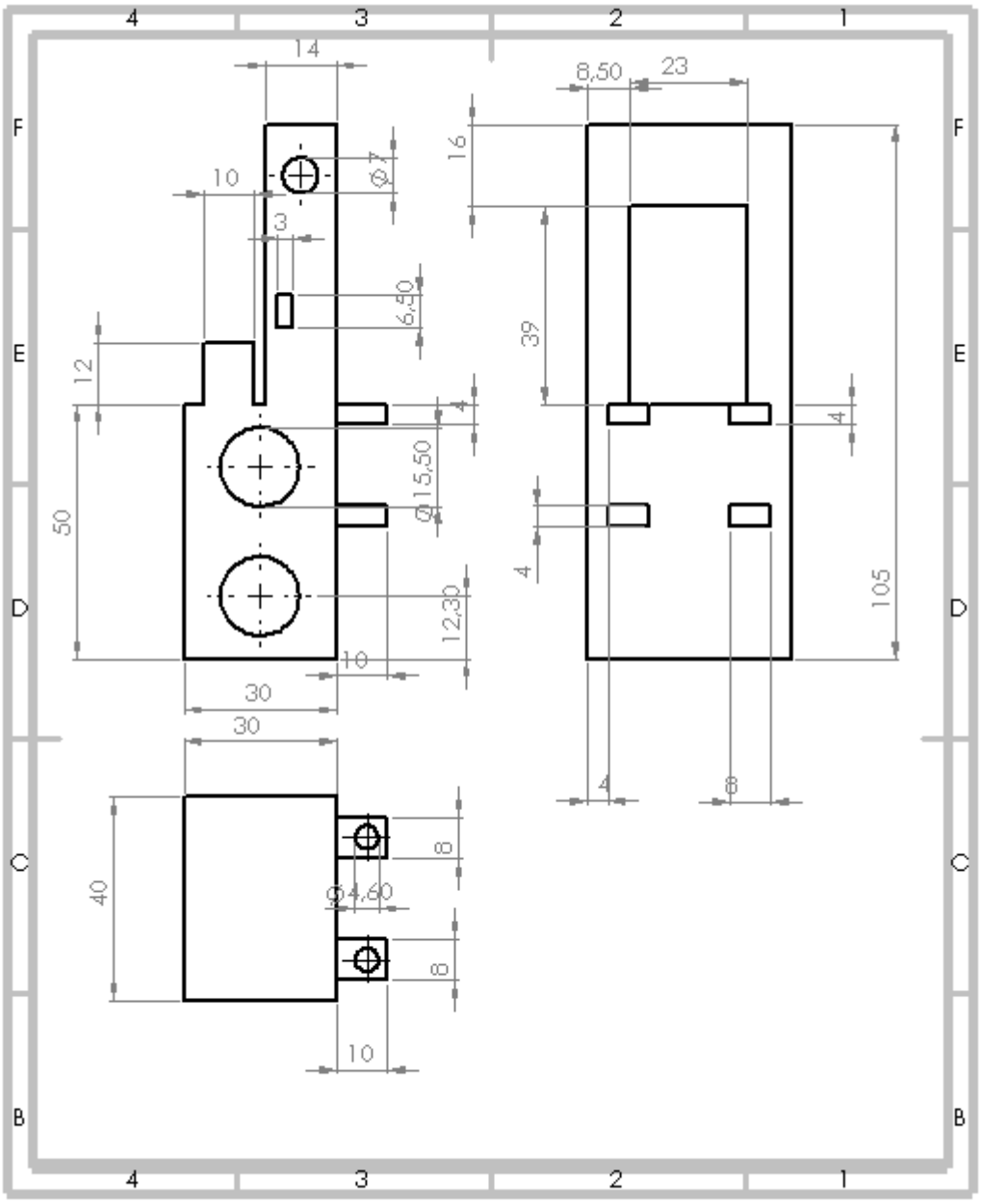
```
128             for i in range(l):
129                 self.x.append(liste_x[l - 1 - i] + x0)
130                 self.y.append(liste_y[l - 1 - i] + y0)
131 
132         def writeFour(self, x0, y0):
133             liste_x, liste_y = [], []
134             x, y = self.L / 12, self.L / 2
135     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

```
        liste_x, liste_y = [], []
        while x < - self.L / 6:
            try:
                x = - math.sqrt(self.L ** 2 / 16 - (y + self.L / 4) ** 2)
            except ValueError:
                x = 0
                y = 0
            liste_x.append(x)
            liste_y.append(y)
            y -= self.step
        while x <= 0:
            y = - self.L / 4 - math.sqrt(self.L ** 2 / 16 - x ** 2)
            liste_x.append(x)
            liste_y.append(y)
            x += self.step
        l = len(liste_x)
        for i in range(l - 1, -1, -1):
            liste_x.append(- liste_x[i])
            liste_y.append(liste_y[i])
        l = len(liste_x)
        for i in range(l - 1, -1, -1):
            liste_x.append(liste_x[i])
            liste_y.append(- self.L / 2 - liste_y[i])
        l = len(liste_x)
        self.append(liste_x, liste_y, x0, y0)

    def writeSeven(self, x0, y0):
        x = - self.L / 4
        y = self.L / 2
        while x <= self.L / 4:
            self.x.append(x + x0)
            self.y.append(y + y0)
            x += self.step
        x = self.L / 4
        while y >= - self.L / 2:
            x = y / 2
            self.x.append(x + x0)
            self.y.append(y + y0)
            y -= self.step
        self.x.append(-self.L / 4 + x0)
        self.y.append(-self.L / 2 + y0)

    def writeEight(self, x0, y0):
        liste_x, liste_y = [], []
        y = 0
        x = 0
        while x < self.L / 6:
            y = self.L / 4 + math.sqrt((self.L / 4) ** 2 - x ** 2)
            liste_x.append(x)
            liste_y.append(y)
            x += self.step
        while y >= self.L / 4:
            x = math.sqrt((self.L / 4) ** 2 - (y - self.L / 4) ** 2)
            liste_x.append(x)
            liste_y.append(y)
            y -= self.step
        l = len(liste_x)
        if self.L / 4 not in liste_x or self.L / 4 not in liste_y:
            liste_x.append(self.L / 4)
            liste_y.append(self.L / 4)
        for i in range(l, -1, -1):
            liste_x.append(liste_x[i])
            liste_y.append(self.L / 2 - liste_y[i])
        l = len(liste_x)
        liste_x.append(0)
        liste_y.append(self.L / 2)
        for i in range(l - 1, -1, -1):
            liste_x.append(- liste_x[i])
            liste_y.append(liste_y[i])
        for i in range(len(liste_x)):
            liste_x.append(liste_x[i])
            liste_y.append(-liste_y[i])
        l = len(liste_x)
```

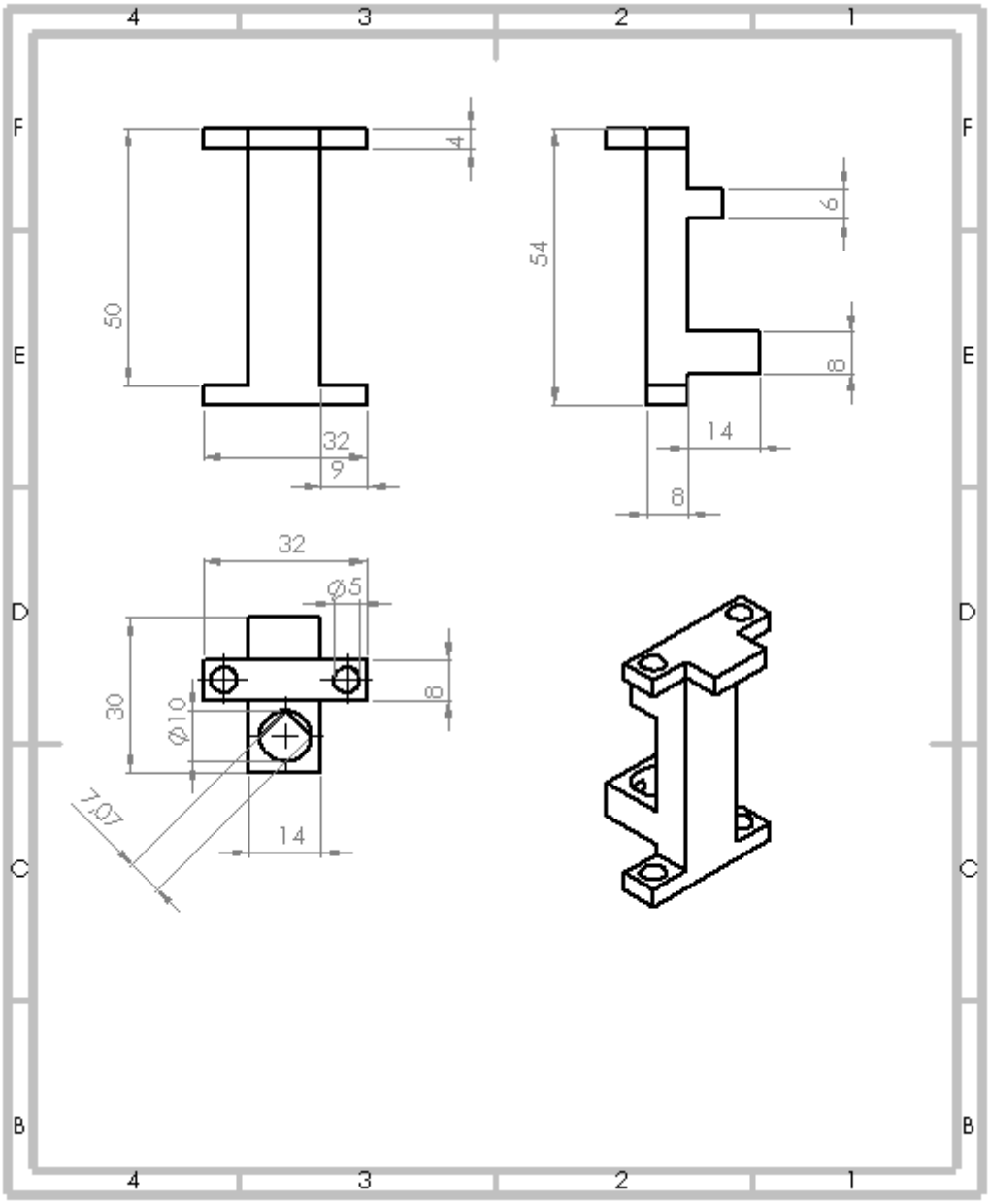
```
274             for i in range(l):
275                 self.x.append(liste_x[l - 1 - i] + x0)
276                 self.y.append(liste_y[l - 1 - i] + y0)
277 
278         def writeNine(self, x0, y0):
279             x, y = - self.L / 5, - self.L / 2
280             while x < - self.L / 12:
281   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

```
                    y -= self.step
        else:
            a = (y1 - y0) / (x1 - x0)
            b = y0 - a * x0
            if x0 < x1:
                while x < x1:
                    y = a * x + b
                    self.x.append(x)
                    self.y.append(y)
                    x += self.step
            else:
                while x > x1:
                    y = a * x + b
                    self.x.append(x)
                    self.y.append(y)
                    x -= self.step

    def writeSudoku(self, sudoku):
        for i in range(10):
            if i % 2 == 0:
                self.writeLine(self.a, self.d + self.ny * i, self.c, self.d + self.ny * i)
            else:
                self.writeLine(self.c, self.d + self.ny * i, self.a, self.d + self.ny * i)
        for i in range(10):
            if i % 2 == 0:
                self.writeLine(self.a + self.nx * (9 - i), self.b, self.a + self.nx * (9 - i), self.d)
            else:
                self.writeLine(self.a + self.nx * (9 - i), self.d, self.a + self.nx * (9 - i), self.b)
        for i in range(9):
            for j in range(9):
                self.writeNumbers(sudoku[i][j], self.x0 + self.nx * j,
                                  self.y0 + self.ny * (8 - i))
        points = []
        for i in range(len(self.x)):
            points.append((self.x[i], self.y[i]))
        return points

    def writeAllNumbers(self):
        for i in range(10):
            self.writeNumbers(i, 4 / 5 * i, 0)

    def write(self, linked=False):
        if linked: plt.plot(self.x, self.y, 'r', linewidth=2)
        else: plt.scatter(self.x, self.y, c='red', s=8)
        plt.grid(True)
        plt.axis('equal')
        plt.axis('off')
        plt.show()


if __name__ == "__main__":
    w = Write()
    sudoku = np.array([[0, 0, 0, 0, 0, 0, 0, 1, 2],
                       [0, 0, 0, 0, 0, 0, 0, 0, 3],
                       [0, 0, 2, 3, 0, 0, 4, 0, 0],
                       [0, 0, 1, 8, 0, 0, 0, 0, 5],
                       [0, 6, 0, 0, 7, 0, 8, 0, 0],
                       [0, 0, 0, 0, 0, 9, 0, 0, 0],
                       [0, 0, 8, 5, 0, 0, 0, 0, 0],
                       [9, 0, 0, 0, 4, 0, 5, 0, 0],
                       [4, 7, 0, 0, 0, 6, 0, 0, 0]])
    print(w.writeSudoku(sudoku))
    w.write(True)
```

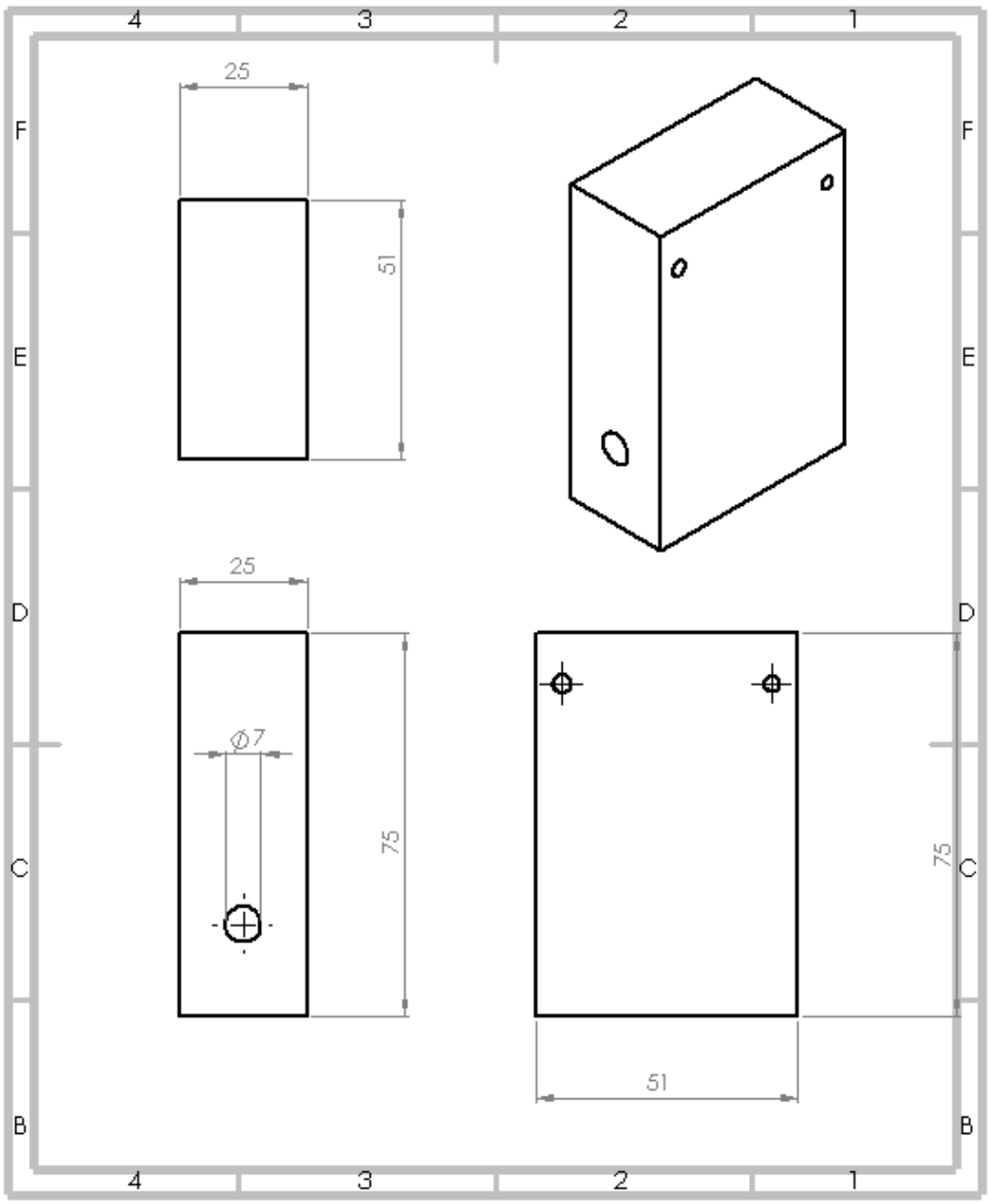
Mise en plan du chariot



Mise en plan du support du stylo



Mise en plan du lien chariot/caméra



Mise en plan du support de la caméra