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

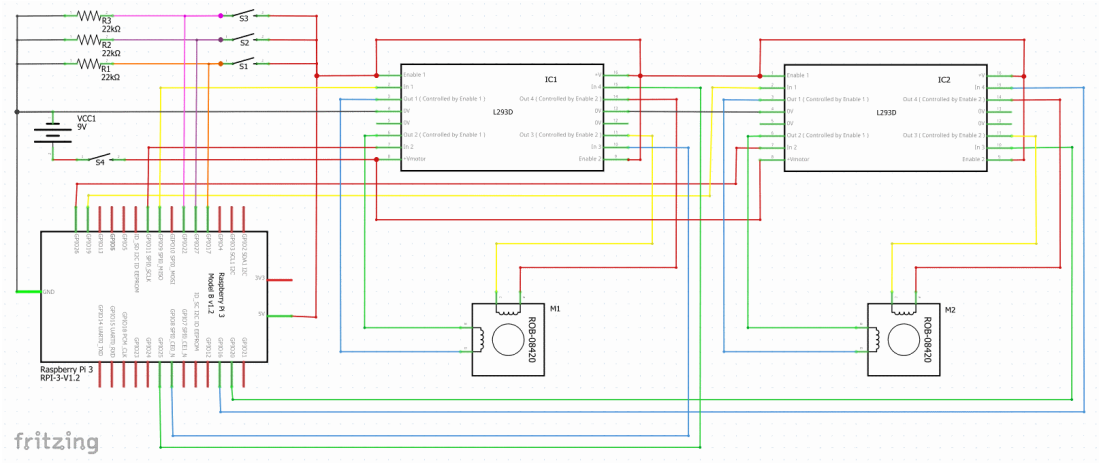


FIGURE 2.1 – Schéma électrique réalisé avec *Fritzing*

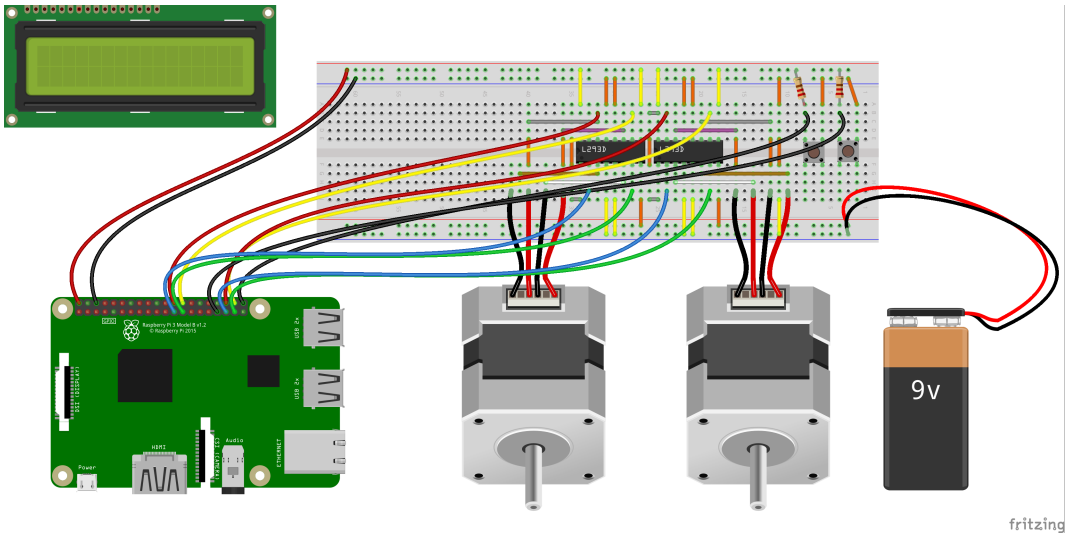


FIGURE 2.2 – Premier montage

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.

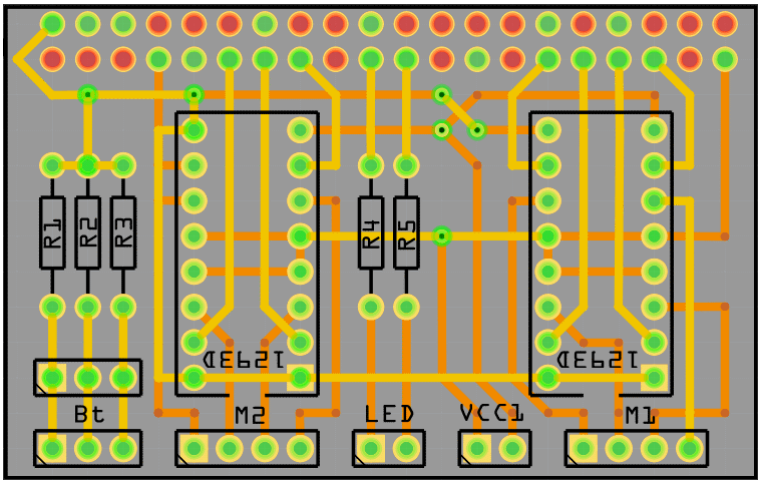


FIGURE 2.3 – Circuit imprimé réalisé avec *Fritzing*

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.

L'écriture se fera en coordonnées polaires et non pas en coordonnées cartésiennes pour les raisons suivantes :

- un système en coordonnées cartésiennes aurait été trop encombrant et un moins "portable" (dans un contexte notamment d'optimalité, comme exigé par le programme)
- c'est un défi technique de fabriquer un bras de ce type en coordonnées polaires

Ainsi, un moteur pas-à-pas assurera la rotation θ du bras. Ce moteur s'auto-entrainera autour d'une roue dentée de 128 dents comme celle-ci et un roulement permettra la rotation de l'entièreté du bras :

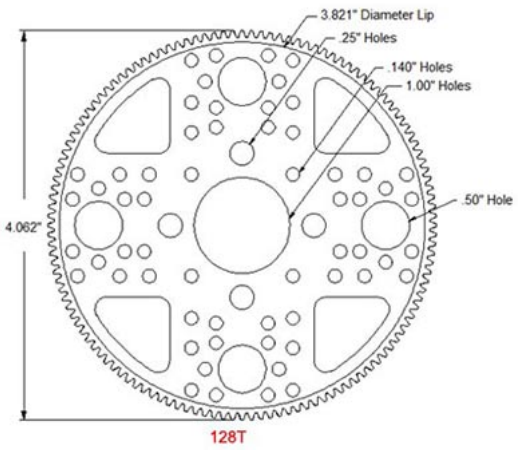


FIGURE 3.1 – Schéma de la roue dentée

De même la translation r a été assurée par un deuxième moteur pas-à-pas et un système courroie/poulie. Cette translation s'est faite le long de deux axes de 272 mm et de 8 mm de diamètre.

3.1 Difficultés rencontrées et solutions mises en place

Nous avons notamment rencontré des difficultés concernant le roulement utilisé. En effet, nous avions au préalable utilisé un roulement à billes comme celui-ci :

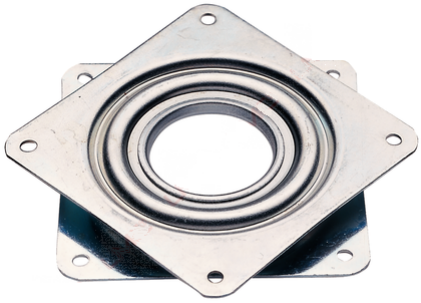


FIGURE 3.2 – Roulement défectueux

Malheureusement, ce roulement tournait très mal et saccadait. De plus, du jeu était présent. Après quelques heures de recherche, nous avons donc choisi le roulement à billes suivant, qui tourne très bien mais qui tout de même possède lui aussi un peu de jeu :

FIGURE 3.3 – Roulement finalement choisi

3.2 Conception du chariot en 3D

Nous avons pris la décision de concevoir certaines pièces de notre projet en 3D, celle-ci nécessitant de la précision. Le logiciel professionnel de modélisation SolidWorks a été utilisé pour développer ces pièces.

Les mises en plan des pièces composant le chariot sont disponibles en annexe **??** à partir de la page **??**.

La partie principale du chariot devait pouvoir accueillir un servomoteur, permettant l'abaissement et le soulèvement du stylo :

Ce stylo sera fixé sur la pièce présentée en page**??** dont l'impression 3D est montrée ci-dessous :

L'ensemble chariot/support du stylo imprimé en 3D est alors montrée ci-dessous :

Le chariot permet, de façon modulaire, d'accueillir un support caméra se divisant en deux pièces présentées en page **??** et en page **??**.

Chapitre 4

Informatique

4.1 Présentation globale

Tous les algorithmes développés dans le cadre de ce projet sont disponibles en annexe [1]. Ils ont été, pour la plupart, développés sous Python 3, les autres sous 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 autres 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, permettant ainsi de tester chacun des scripts très facilement.

4.2 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*. Les grilles les plus difficiles peuvent imposer au joueur de faire un choix entre plusieurs possibilités pour espérer mener à son terme la résolution. C'est sur ce constat que ce base la méthode dite de *backtracking* (ou *retour sur trace*), qui permet de résoudre toute grille de sudoku, même si celle-ci n'est pas valide et possède plusieurs solutions.

Inclusion

1. Cf **??**

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 pouvant théoriquement convenir, puis on liste les nouvelles possibilités des autres cases en fonction du chiffre précédemment ajouté puis on passe à la case suivante. S'il n'y a plus aucune possibilité, on revient sur nos pas et on change le chiffre qu'on venait d'ajouter en une autre possibilité. Si tout les chiffres possibles ont été testé, on revient de nouveau sur nos pas autant de fois que nécessaire, jusqu'à avoir compéter la totalité de la grille. Cette méthode ne peut pas être mise en oeuvre par un joueur, dans la mesure où elle lui demanderait énormément de temps, et ne présenterait pour lui aucun intérêt. Cependant, elle peut être indispensable pour les grilles diaboliques lorsque le joueur doit faire un choix, cette méthode est alors indispensable.

Partant de ce constat, nous avons tout d'abord implémenté la méthode de *backtracking*, ainsi, il nous était possible de résoudre toutes les grilles possibles. À l'origine, notre algorithme parcourait la grille dans un ordre prédéfini, à savoir de haut en bas et de gauche à droite. Cependant, il pouvait arriver que pour certaines grilles, ce système n'était pas tout à fait optimale. C'est pourquoi, nous avons changé ce système, de sorte que l'algorithme commence par les cases présentant le moins de possibilités, permettant dans le cas général de diminuer le temps de résolution. Cependant, dans le pire des cas, cette solution ne présentait aucune différence de temps de calculs.

4.3 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 :

– Reconnaître les chiffres dans une grille du sudoku

– Déterminer la position spatiale de la grille

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

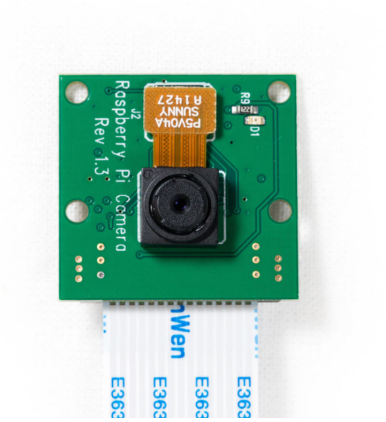


FIGURE 4.1 – Caméra Raspberry Pi V2

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

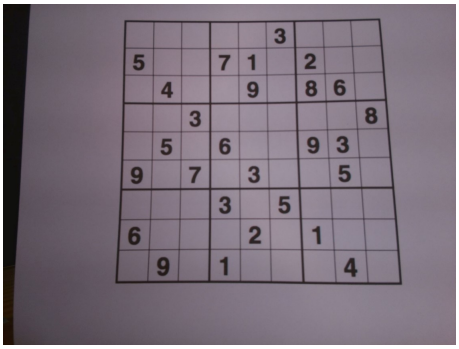


FIGURE 4.2 – Exemple de grille de sudoku

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 :

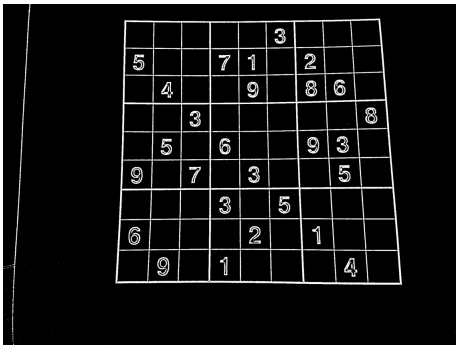


FIGURE 4.3 – Exemple de grille "seuillée"

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.

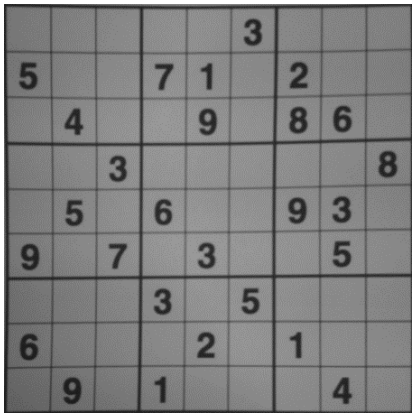


FIGURE 4.4 – Exemple de grille découpée sans perspective

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

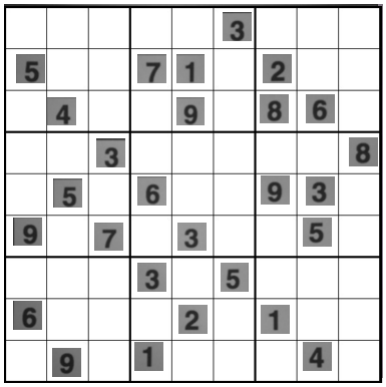


FIGURE 4.5 – Exemple de grille où chaque chiffre a été découpé

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

«««< HEAD

4.4 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* (en français : *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.
======= »»»> 947ccc4b7b6b245dd59ccde8c0416194b54cb2d4

4.5 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.

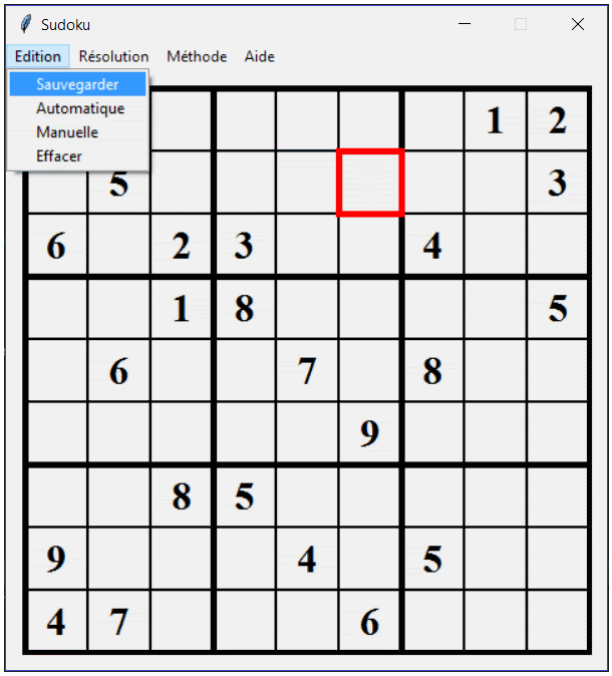


FIGURE 4.7 – Interface graphique affichant le menu Edition

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.

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

4.6 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

2. cf section **??** à la page **??**

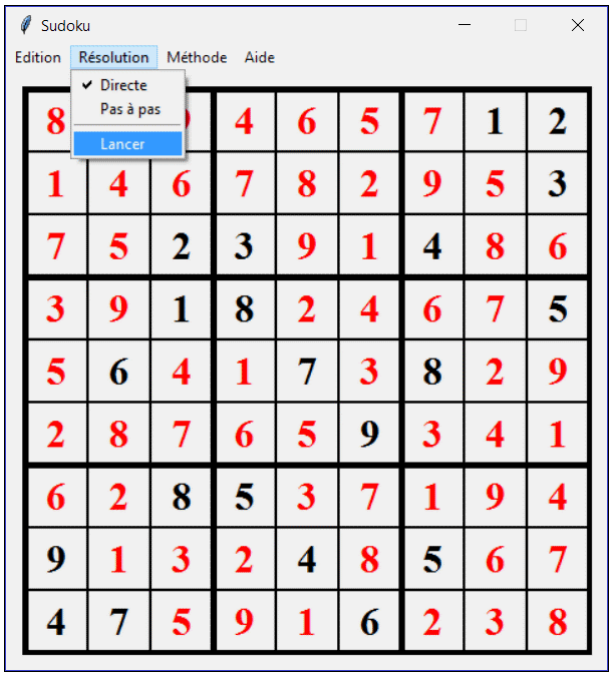


FIGURE 4.8 – Interface graphique affichant le menu Édition

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.

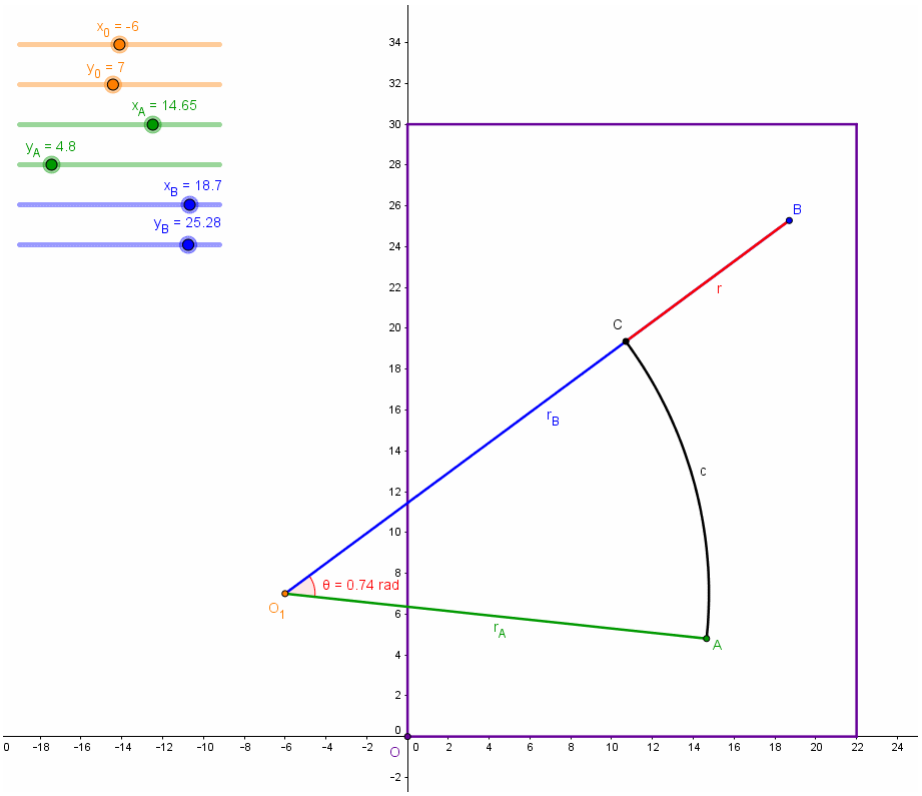


FIGURE 4.9 – Changement de repères réalisé avec *Geogebra*

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 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.

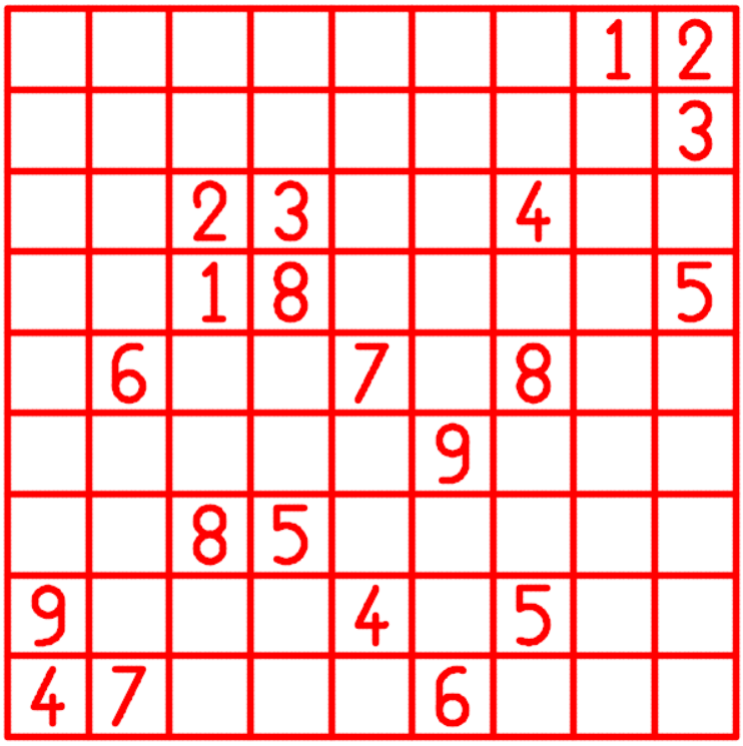


FIGURE 4.11 – Grille de sudoku en point par point tracée avec *MatplotLib*

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.

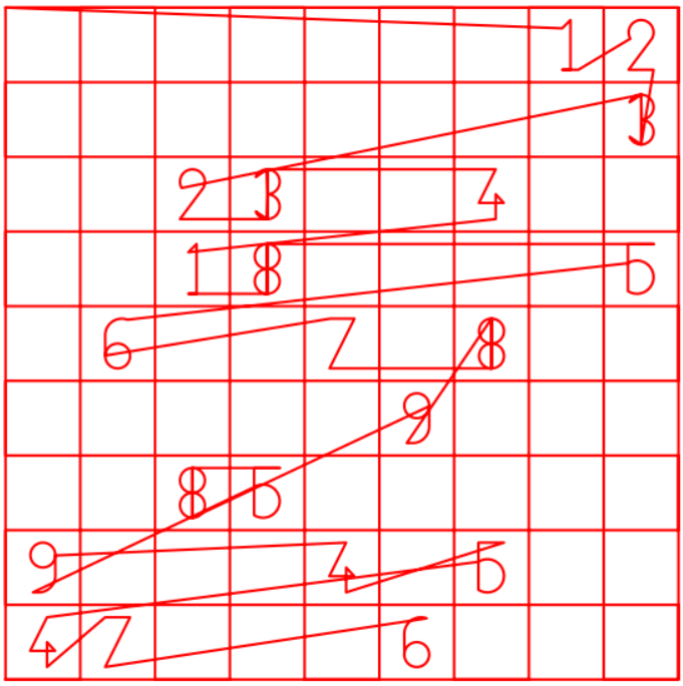


FIGURE 4.12 – Grille de sudoku tracée avec *MatplotLib*

Conclusion

Tous les scipts et ce dossier sont disponibles sur GitHub [3]

3. 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.

Script principal

Script de résolution des sudokus

Script de gestion de la caméra

Script d'affichage du sudoku

Script de gestion des moteurs pas-à-pas

Script permettant l'écriture d'un sudoku

«««< HEAD

```
#!/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)
```

```
59              liste_y.append(y)
60              y += self.step
61          liste_x.append(- self.L / 4)
62          liste_y.append(self.L / 4)
63          while x <= 0:
64              y = self.L / 4 + math.sqrt((self.L / 4) ** 2 - x ** 2)
65              liste_x.append(x)
66              liste_y.append(y)
67              x += self.step
68          liste_x.append(0)
69          liste_y.append(self.L / 2)
70          l = len(liste_x)
71          for i in range(l - 1, -1, -1):
72              liste_x.append(- liste_x[i])
73              liste_y.append(liste_y[i])
74          x, y = liste_x[-1], liste_y[-1]
75          a = (5 * self.L / 8) / (x + self.L / 4)
76          b = self.L / 2 * (a / 2 - 1)
77          x -= self.step
78          while x > - self.L / 4:
79              y = a * x + b
80              liste_x.append(x)
81              liste_y.append(y)
82              x -= self.step
83          x = - self.L / 4
84          y = - self.L / 2
85          while x < self.L / 4:
86              liste_x.append(x)
87              liste_y.append(y)
88              x += self.step
89          liste_x.append(self.L / 4)
90          liste_y.append(y)
91          self.append(liste_x, liste_y, x0, y0)
92
93      def writeThree(self, x0, y0):
94          liste_x, liste_y = [], []
95          y = 0
96          x = 0
97          while x < self.L / 6:
98              y = self.L / 4 + math.sqrt((self.L / 4) ** 2 - x ** 2)
99              liste_x.append(x)
100             liste_y.append(y)
101             x += self.step
102         while y >= self.L / 4:
103             x = math.sqrt((self.L / 4) ** 2 - (y - self.L / 4) ** 2)
104             liste_x.append(x)
105             liste_y.append(y)
106             y -= self.step
107         l = len(liste_x)
108         if self.L / 4 not in liste_x or self.L / 4 not in liste_y:
109             liste_x.append(self.L / 4)
110             liste_y.append(self.L / 4)
111         liste_x.append(0)
112         liste_y.append(0)
113         for i in range(l, -1, -1):
114             if liste_x[i] > 0:
115                 liste_x.append(liste_x[i])
116                 liste_y.append(self.L / 2 - liste_y[i])
117         l = len(liste_x)
118         liste_x.append(0)
119         liste_y.append(self.L / 2)
120         for i in range(l - 1, -1, -1):
121             if liste_y[i] > self.L / 3:
122                 liste_x.append(- liste_x[i])
123                 liste_y.append(liste_y[i])
124         for i in range(len(liste_x)):
125             liste_x.append(liste_x[i])
126             liste_y.append(-liste_y[i])
127         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
201         liste_x, liste_y = [], []
202         while x < - self.L / 6:
203             try:
204                 x = - math.sqrt(self.L ** 2 / 16 - (y + self.L / 4) ** 2)
```

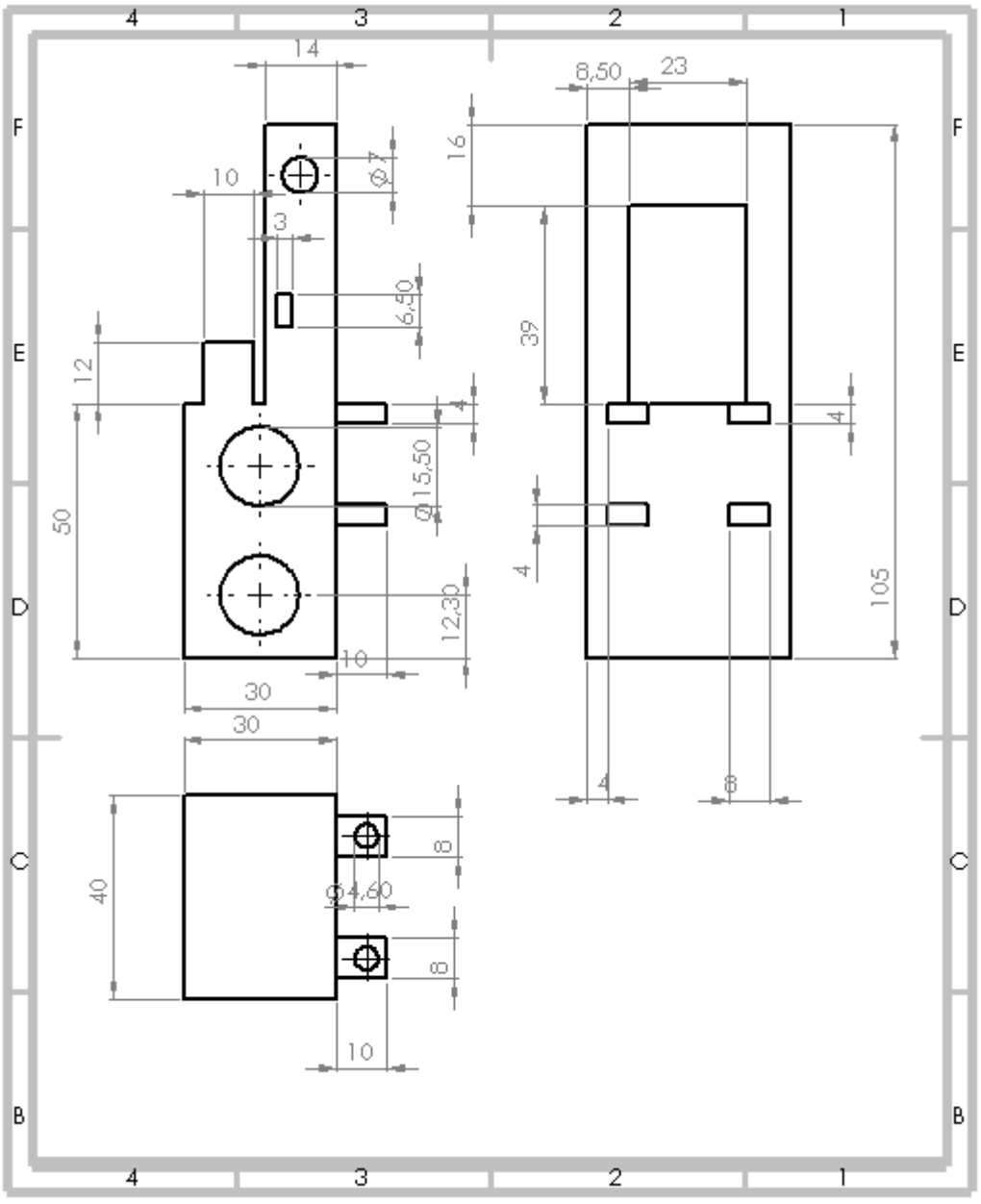
```
            except ValueError:
                x = 0
                y = 0
            liste_x.append(x)
            liste_y.append(y)
            y -= self.step
        while x <= 0:
            y = - self.L / 4 - math.sqrt(self.L ** 2 / 16 - x ** 2)
            liste_x.append(x)
            liste_y.append(y)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

```

```
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)
347                     y -= self.step
348         else:
349             a = (y1 - y0) / (x1 - x0)
350             b = y0 - a * x0
```

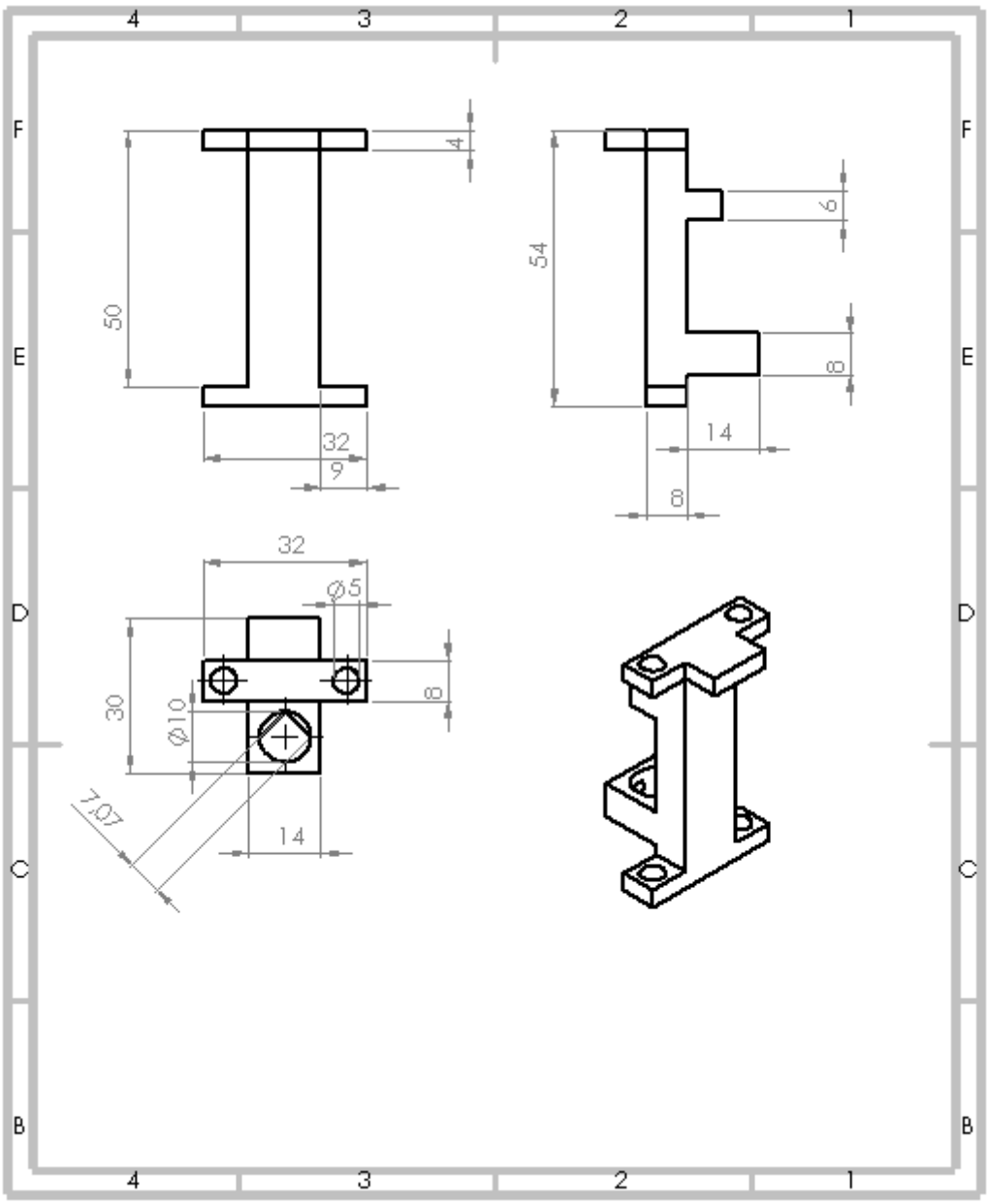
```
            if x0 < x1:
                while x < x1:
                    y = a * x + b
                    self.x.append(x)
                    self.y.append(y)
                    x += self.step
            else:
                while x > x1:
                    y = a * x + b
                 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

======= »»»> 947ccc4b7b6b245dd59ccde8c0416194b54cb2d4

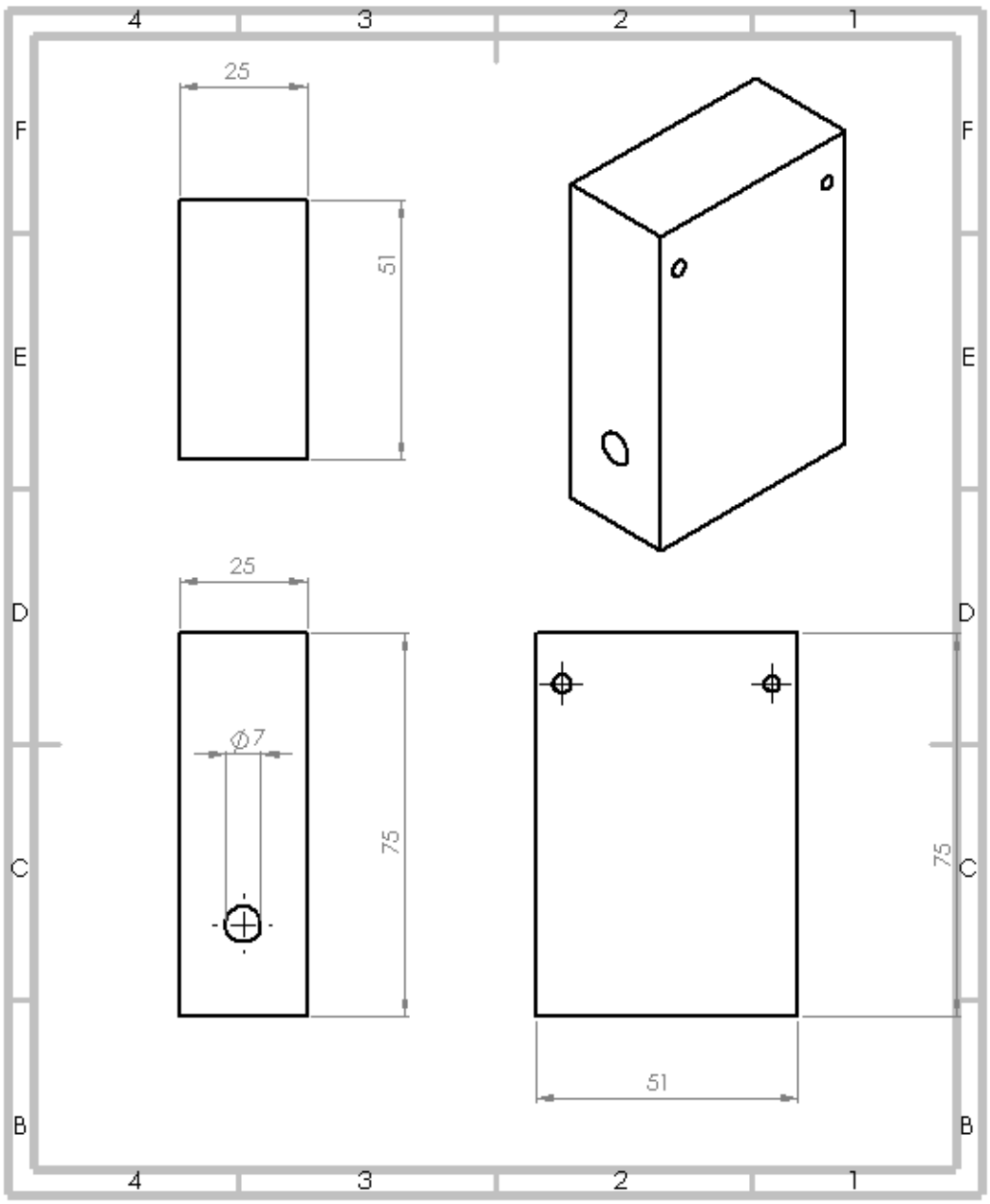
Mise en plan du chariot



Mise en plan du support du stylo



Mise en plan du lien chariot/caméra



Mise en plan du support de la caméra