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 **1**

1 Présentation et faits sur le sudoku **3**

2 Mécanique **4**
2.1 Choix concernant la structure du bras 4
2.2 Étude cinématique du mécanisme 5
2.3 Difficultés rencontrées et solutions mises en place 7
2.4 Conception du chariot en 3D 7

3 Électronique **9**
3.1 Moteurs pas-à-pas 9
3.2 Drivers L293D 11
3.3 Schéma électrique 12
3.4 Premiers montages 13
3.5 Circuit imprimé 13
3.6 Écran LCD 14

4 Informatique **15**
4.1 Présentation globale 15
4.2 Reconnaissance du sudoku 15
4.3 Résolution du sudoku 18
4.4 Interface graphique 21
4.5 Contrôle des moteurs et écriture 22

Conclusion **27**

Remerciements **28**

Introduction

[1]

1. Sauf mention contraire, les images et figures ont été réalisées par nos soins.

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 : $n = 3$. On appelle section, une ligne, une colonne ou une région

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

– Chaque section 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

Mécanique

2.1 Choix concernant la structure du bras

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 :

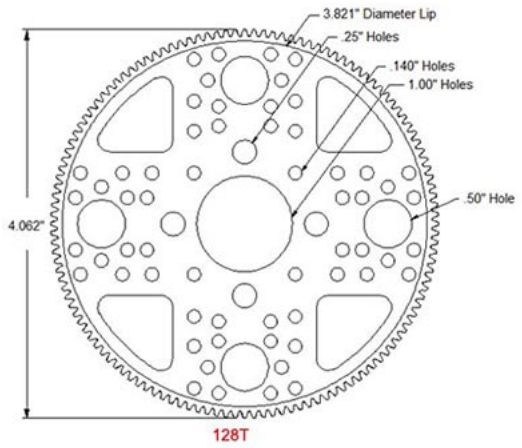

FIGURE 2.1 – Schéma de la roue dentée

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.

2.2 Étude cinématique du mécanisme

La pièce 0 correspondra par la suite au bâti. La base du bras et le bâti seront supposés en liaison encastrement, alors qu'ils sont en réalité en liaison appui-plan, cependant les forces de frottements entre ces deux pièces sont telles quelles sont presque immobiles l'une par rapport à l'autre. La pièce 1 correspond à la structure du bras, reposant sur la roue dentée. La pièce 2 symbolise le chariot et la pièce 3 le stylo.

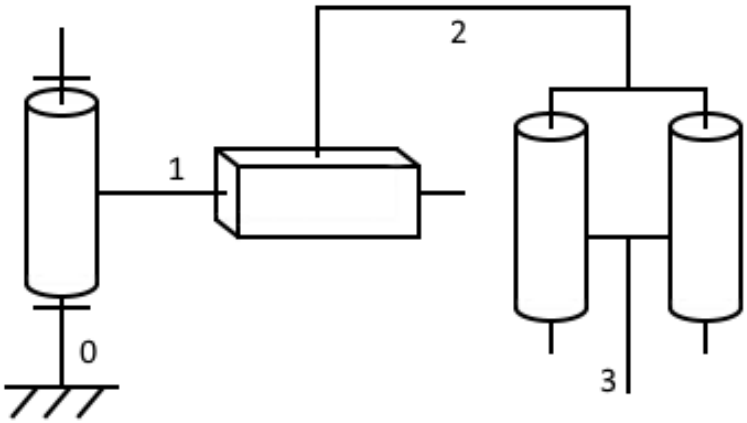

FIGURE 2.2 – Schéma cinématique du mécanisme choisi

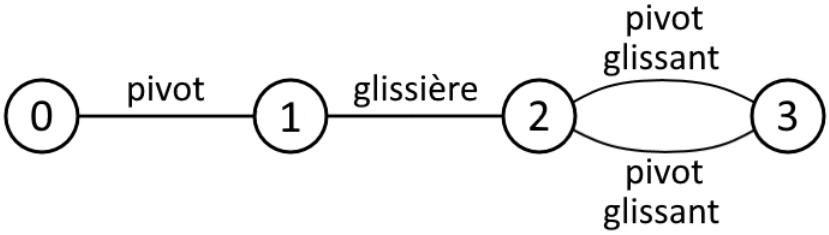

FIGURE 2.3 – Graphe de structure du mécanisme choisi

Les deux liaisons en parallèle entre 2 et 3 peuvent être remplacés par une liaison équivalent ede torseur $\{T_c(2/3)\}$ devant vérifier l'équation suivante : :

$$\{T_c(2/3)\}_B = \{T'_c(2/3)\}_B = \{T''_c(2/3)\}_B$$

On notera les liaisons entre les pièces 2 et 3 $L'_{2/3}$ et $L''_{2/3}$. On a alors :

$$\{T'_c(2/3)\} = \begin{matrix} \\ \forall P \in (B,\vec{z}) \end{matrix} \left\{ \begin{array}{cc} 0 & 0 \\ 0 & 0 \\ \gamma'_{2/3} & w'_{2/3} \end{array} \right\}_{x,y,z)}$$

$$\{T''_c(2/3)\} = \begin{matrix} \\ \forall P \in (C,\vec{z}) \end{matrix} \left\{ \begin{array}{cc} 0 & 0 \\ 0 & 0 \\ \gamma''_{2/3} & w''_{2/3} \end{array} \right\}_{(x,y,z)}$$

$$\begin{aligned}\text{Or } \overrightarrow{V}(B \in 2/3) &= \overrightarrow{V}(C \in 2/3) + \overrightarrow{\Omega}(2/3) \wedge \overrightarrow{CB} \\ &= w''_{2/3}\overrightarrow{z} + \gamma''_{2/3}\overrightarrow{z} \wedge BC\overrightarrow{x} \\ &= w''_{2/3}\overrightarrow{z} + \gamma''_{2/3}BC\overrightarrow{y}\end{aligned}$$

$$\text{On a donc } \{T''_c(2/3)\} = \begin{matrix}\\ B\end{matrix}\left\{\begin{array}{cc} 0 & 0 \\ 0 & \gamma''_{2/3}BC \\ \gamma''_{2/3} & w''_{2/3} \end{array}\right\}_{(x,y,z)}$$

$$\text{Donc } \{T_c(2/3)\} = \begin{matrix}\\ B\end{matrix}\left\{\begin{array}{cc} 0 & 0 \\ 0 & 0 \\ \gamma'_{2/3} & w'_{2/3} \end{array}\right\}_{x,y,z)} = \begin{matrix}\\ B\end{matrix}\left\{\begin{array}{cc} 0 & 0 \\ 0 & \gamma''_{2/3}BC \\ \gamma''_{2/3} & w''_{2/3} \end{array}\right\}_{(x,y,z)}$$

$$\text{D'où le système d'équation : } \left\{\begin{array}{l} 0 = 0 \\ 0 = 0 \\ \gamma'_{2/3} = \gamma''_{2/3} \\ 0 = 0 \\ \gamma''_{2/3} = 0 \\ w'_{2/3} = w''_{2/3} \end{array}\right.$$

$$\text{On en déduit } \{T_c(2/3)\} = \begin{matrix}\\ B\end{matrix}\left\{\begin{array}{cc} 0 & 0 \\ 0 & 0 \\ 0 & w'_{2/3} \end{array}\right\}_{x,y,z)}$$

La liaison équivalente est donc une liaison glissière d'axe z, ce qui permet des déplacements verticaux du stylo, il peut ainsi être mis en contact avec la feuille. On a alors le schéma de la figure 2.4.

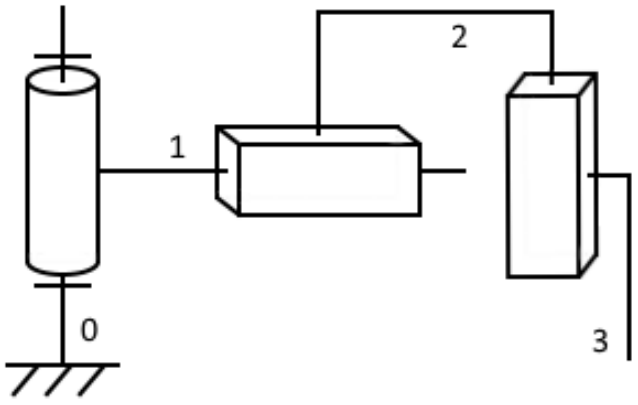

FIGURE 2.4 – Schéma cinématique équivalent au mécanisme choisi

2.3 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 :

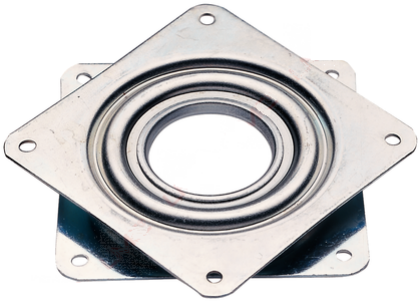

FIGURE 2.5 – Roulement défectueux

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 2.6 – Roulement finalement choisi

2.4 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, développé par DassaultSystèmes, 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 3

Électronique

3.1 Moteurs pas-à-pas

Nous avons utilisé 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.

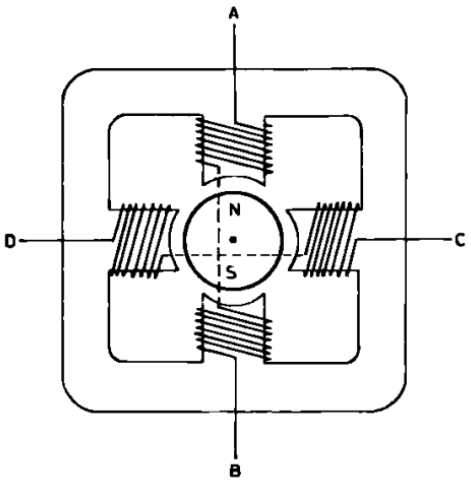

FIGURE 3.1 – Schéma simplifiée d'un moteur pas-à-pas [1]

Les moteurs que nous avons utilisés sont constitués de deux bobines, repliées tout autour de la partie centrale du moteur, permettant des petits déplacements de la partie mobile. Celles-ci sont divisées en deux sur la figure 3.1. La bobine 1 a pour extrémités A et B ; la bobine 2 C et D. Ces bobines, lorsqu'elles

1. http://eskimon.fr/290-arduino-603-petits-pas-le-moteur-pas-pas

sont parcourues par un courant i, génèrent un champ magnétique. Ce champ magnétique a pour conséquence d'entrainer la rotation de la partie mobile du moteur : le rotor, représenté par le cercle centrale N/S. Celui-ci est composé d'un aimant qui a tendance a s'aligner sur le champ magnétique créé par les bobines. Les bobines sont enroulées autour d'un fer doux, généralement de la ferrite afin de mieux conduire le lignes de champs.

On distingue deux types de fonctionnement pour ces moteurs bipolaires, un fonctionnement à demi-pas et un fonctionnement à pas complet. Nous avons retenu ce dernier système car il permettait d'obtenir une vitesse et un couple constants lors de la rotation des moteurs. Comme le montre la figure 3.2, à chaque séquence, une seule bobine suffirait à mettre en mouvement le moteur, cependant nous avons choisi d'alimenter tout de même l'autre afin d'obtenir un couple maximal (cf figure 3.3).

Séquence	Bobine 1		Bobine 2	
	A	B	C	D
1	1	0	-	-
2	-	-	0	1
3	0	1	-	-
4	-	-	1	0
5	1	0	-	-

TABLE 3.1 – Séquences d'alimentation des bobines avec un couple minimal

Séquence	Bobine 1		Bobine 2	
	A	B	C	D
1	1	0	0	1
2	0	1	0	1
3	0	1	1	0
4	1	0	1	0
5	1	0	0	1

TABLE 3.2 – Séquences d'alimentation des bobines avec un couple maximal

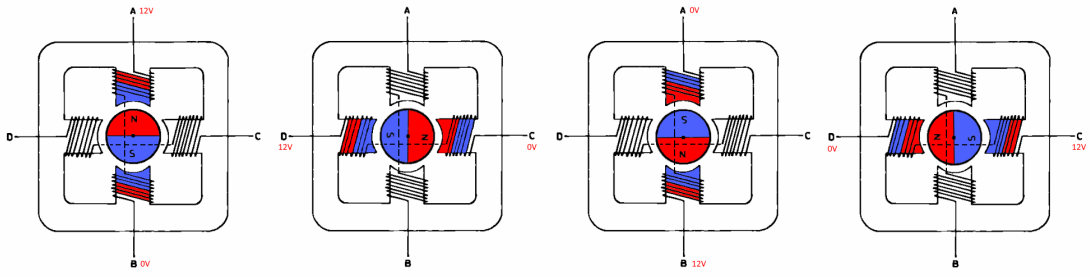

FIGURE 3.2 – Fonctionnement à pas complet avec un couple minimal [2]

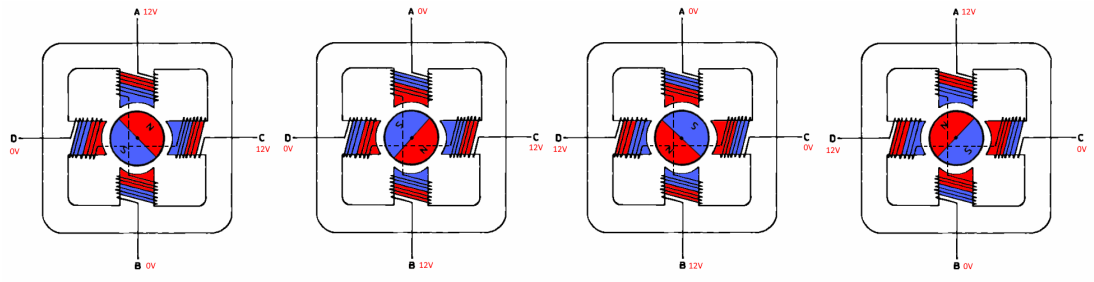

FIGURE 3.3 – Fonctionnement à pas complet avec un couple maximal [3]

Une autre différence entre l'alimentation d'une ou deux bobines réside dans la position de l'axe des deux moteurs, la position de celui-ci est décalé d'un demi-pas. Le fonctionnement en demi-pas se base ainsi sur la combinaison de ces deux méthodes, ce qui entraîne un couple variable mais permet des déplacements plus précis.

3.2 Drivers L293D

Pour piloter les moteurs, nous utilisons des drivers L293D. Ces drivers présentent les avantages suivants :

- ils permettent l'utilisation d'une source de tension externe, la *Raspberry* ne pouvant fournir que du 5V ;
- ils intègrent deux ponts en H permettant d'inverser le courant circulant dans les bobines ;
- ils permettent d'éviter la création de courants induits dans la *Raspberry*, du faits des bobines, pouvant potentiellement l'endommager ou la détruire.

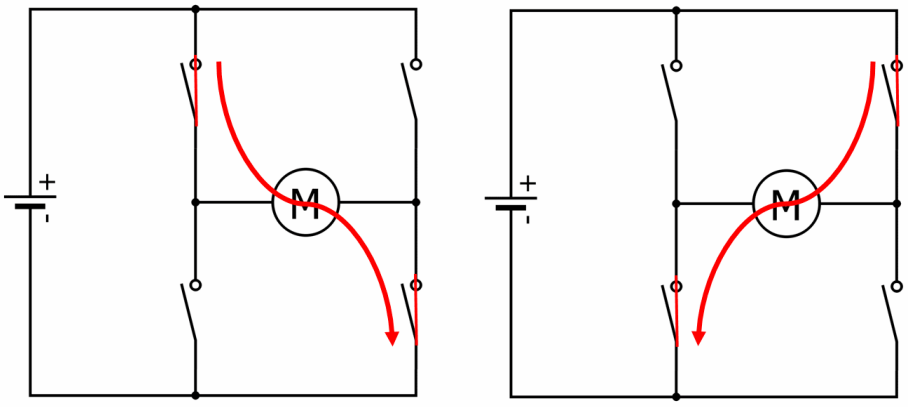

FIGURE 3.4 – Utilité du pont en H, réalisé avec *QElectroTech*

2. Une animation du pas complet avec couple minimal est disponible sur `www.github.com/alphter/Sudoku-Plotter/blob/master/pictures/motor_min1.gif`
3. Une animation du pas complet avec couple maximal est disponible sur `www.github.com/alphter/Sudoku-Plotter/blob/master/pictures/motor_max1.gif`

Les ponts en H permettent l'alimentation d'une bobine dans les deux sens, ils permettent également de freiner le moteur en le court-circuitant. Celui-ci est constitué de quatre interrupteurs, conformément à la figure 3.5. Le moteur M représente une bobine. Les ponts en H sont utilisés dans quasiment tous les drivers des moteurs, que ce soit des moteurs continus ou des pas-à-pas. Les interrupteurs sont alors remplacés par des transistors, ce qui permet de les commander par une autre source d'alimentation (par exemple du 5V). Pour éviter la création de courants induits dans les bobines, on place généralement des diodes en parallèle des transistors (ou des interrupteurs).

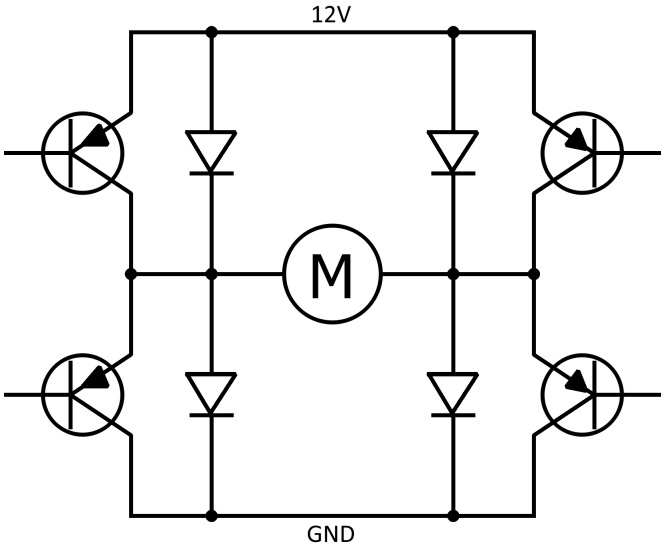

FIGURE 3.5 – Pont en H intégré dans les drivers, réalisé avec *QElectroTech*

Le driver L293D présente l'avantage d'intégrer deux pont en H ainsi que les diodes en parallèle ce qui permet des branchements aisés, sans besoin d'ajout d'autres composants. Comme nos moteurs intègrent deux bobines chacun, nous avons eu besoin de deux L293D, un par moteur.

3.3 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 de la *Raspberry Pi* (notamment du convertisseur de tension).

Les moteurs deux bobines de chaque moteur sont reliées aux 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. La figure 3.6 souligne la présence de deux alimentations différentes, à la fois l'alimentation 5V pour la *Raspberry Pi* et l'alimentation 9V(ou 12V) pour alimenter les moteurs.

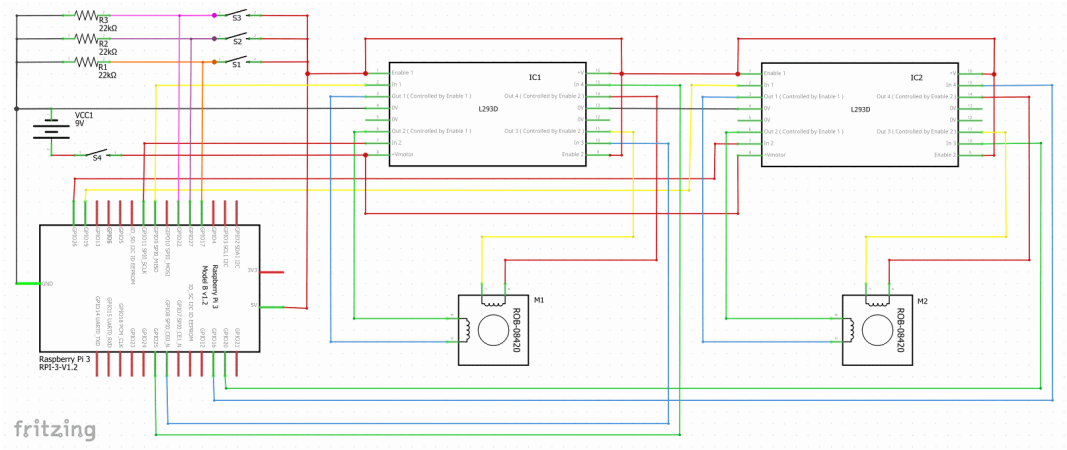

FIGURE 3.6 – Schéma électrique réalisé avec *Fritzing*

3.4 Premiers montages

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

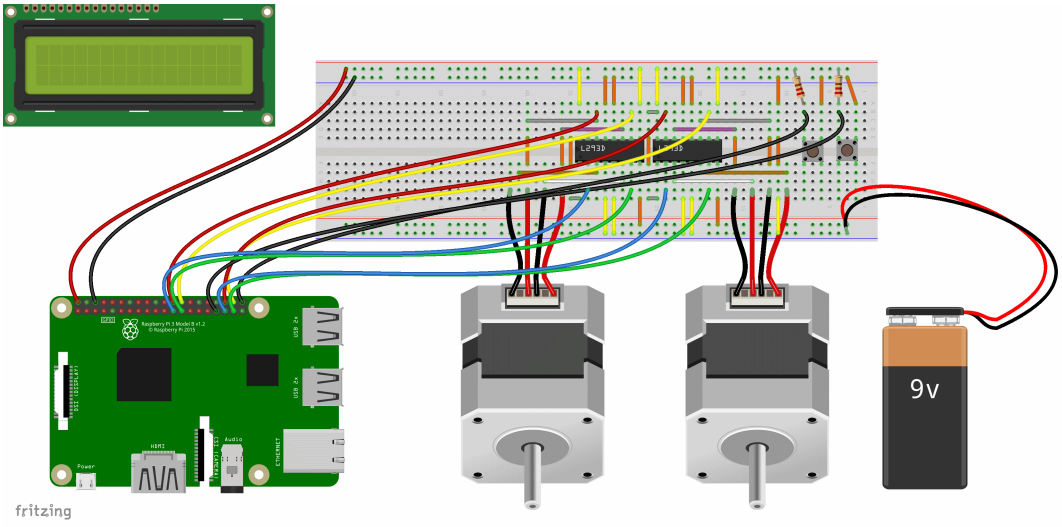

FIGURE 3.7 – Premier montage réalisé avec *Fritzing*

3.5 Circuit imprimé

Pour rendre notre bras mécanique plus compact et ainsi rentrer dans le cadre de l'optimalité, nous avons souhaité remplacer la breadboard par une solution bien plus compacte, à savoir un circuit imprimé. Celui-ci sera enficher directement sur les ports GPIO de la Raspberry Pi. 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 autorisant 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 des deux couches, cela se serait révélé très difficile et aurait sans doute coûté plus cher.

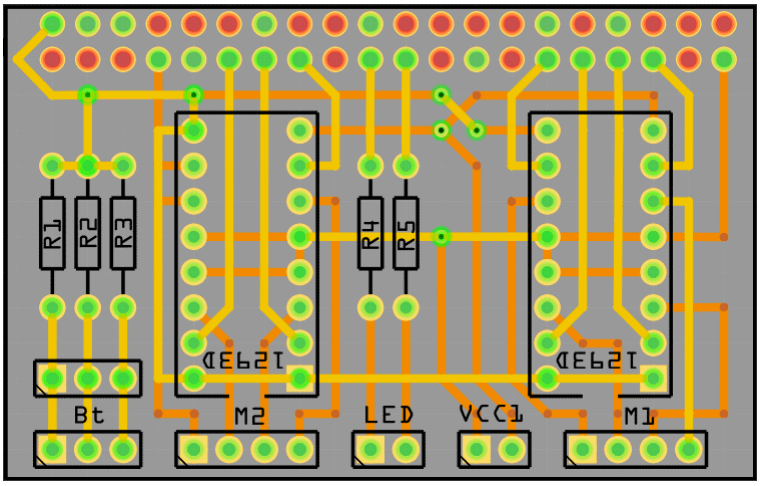

FIGURE 3.8 – Circuit imprimé réalisé avec *Fritzing*

3.6 Écran LCD

Pour rendre le bras mécanique plus 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, ce qui permettait une utilisation facilitée de celui-ci. L'écran utilisé est un écran LCD 2x16, disposant d'un potentiomètre intégré permettant de régler la luminosité de l'écran. Il dispose de quatre sorties, à savoir une sortie 5V, la masse et deux sorties pour envoyer les informations.

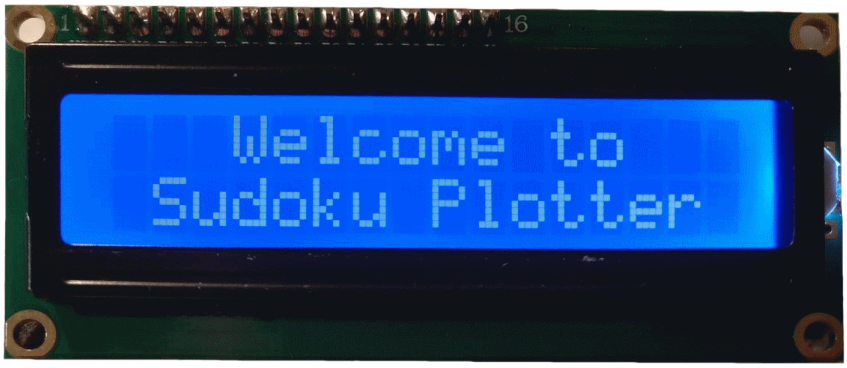

FIGURE 3.9 – Écran LCD affichant le message de bienvenue

Chapitre 4

Informatique

4.1 Présentation globale

Tous les algorithmes développés dans le cadre de ce projet sont disponibles en annexe. Ils ont été, pour la plupart, développés 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 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 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 :

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

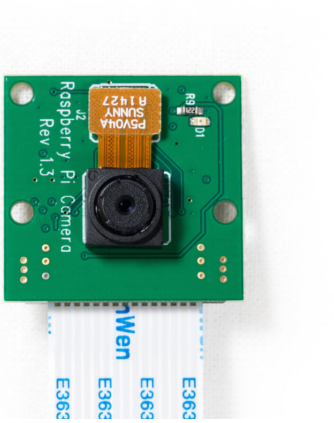

FIGURE 4.1 – Caméra Raspberry Pi V2

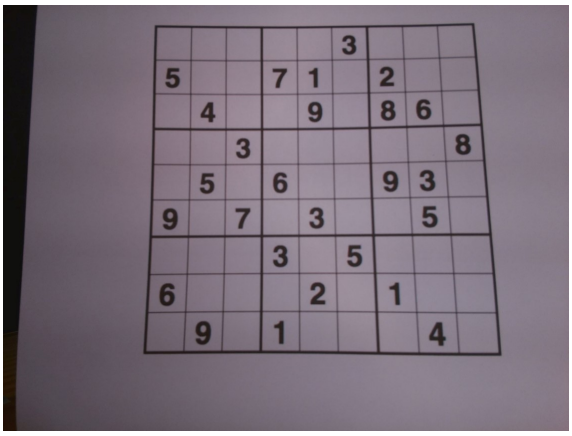

FIGURE 4.2 – Exemple de grille de sudoku

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 :

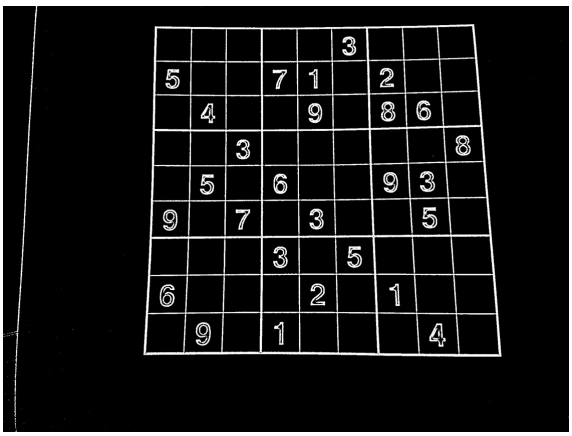

FIGURE 4.3 – Exemple de grille "seuillée"

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.

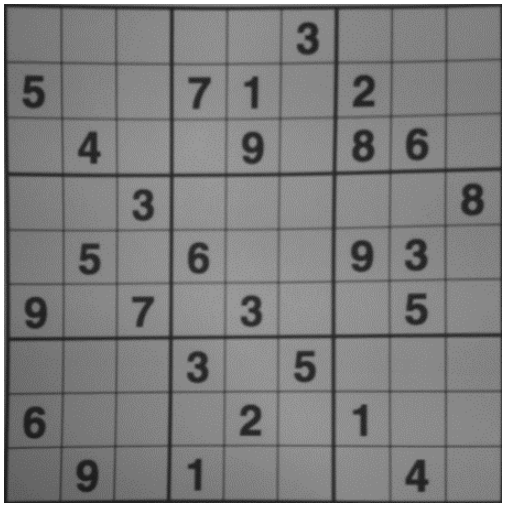

FIGURE 4.4 – Exemple de grille découpée sans perspective

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

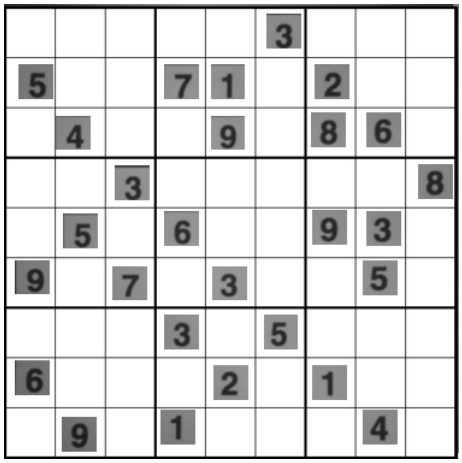

FIGURE 4.5 – Exemple de grille où chaque chiffre a été découpé

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 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. Bien sûr cette liste n'est pas exhaustive et il existe d'autres méthodes de résolution, même si ces méthodes ne sont pas forcément très employées et faciles d'emploi.

Inclusion : un seul candidat pour une case On dénombre pour chaque case vierge les chiffres pouvant être positionnés dans celle-ci. S'il n'y a qu'une seule possibilité alors elle est solution et est donc placée dans la case.

1		2		9		8	6	
4	7							
	5	3						

FIGURE 4.7 – Méthode d'inclusion pour une région

Exclusion : candidat unique pour une section Si pour un chiffre donné, on ne peut le placer dans une section que dans une seule case possible, alors ce chiffre est placé dans cette case

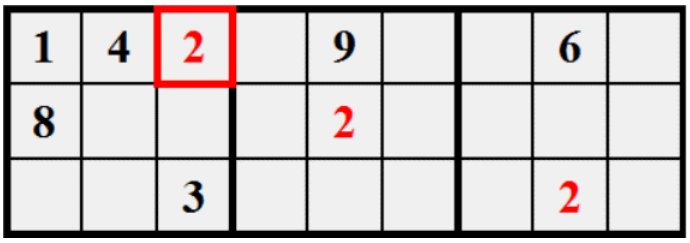

FIGURE 4.8 – Méthode d'exclusion pour une région

Paires Si dans une section donnée, deux cases ne contiennent que deux *candidats identiques*, alors ils ne peuvent être placés que dans ces deux cases, ce qui permet de les éliminer des autres possibilité pour la section

FIGURE 4.9 – Méthode des paires pour une ligne

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

Jumeaux Si dans une section, un chiffre ne peut être placé que sur une ligne (resp. colonne), alors ce chiffre peut être supprimé des possibilités pour le reste de la ligne (resp.colonne). Cette méthode ne permet pas toujours de placer immédiatement celui-ci, mais permet de réduire les possibilités pour placer les autres.

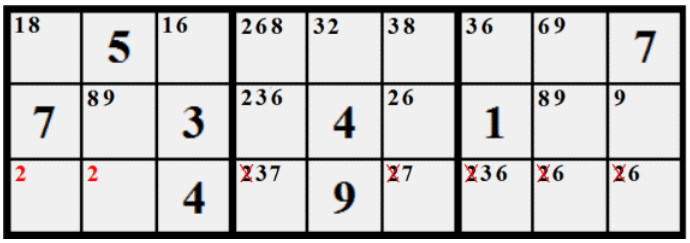

FIGURE 4.10 – Méthode des jumeaux pour une ligne

Backtracking Cette méthode[1] 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.

1. Une animation du *backtracking* est disponible sur : `www.github.com/alphter/Sudoku-Plotter/blob/master/pictures/backtracking.gif`

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 l'avons changé 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 en terme de temps de calculs.

Temps de calculs Au niveau informatique, ces méthodes ne se valent pas dans la mesure où elles sont plus ou moins fiables, efficaces et rapides. L'*inclusion* et l'*exclusion* sont comparables : elles sont rapides et efficaces mais ne permettent pas la résolution complète dans un certain nombre de cas. L'utilisation du *backtracking* se révèle alors efficace en complément des deux précédentes, puisque cette méthode permet toujours de conclure. Les temps de calculs sont comparables pour toutes ces méthodes et sont de l'ordre du centième de secondes [2]. La résolution peut parfois prendre plus d'une seconde mais cette situation ne s'est présentée que très rarement. Cependant, il peut arriver que pour des grilles très particulières, conçues pour tester la rapidité du *backtracking*, la résolution du sudoku prenne bien plus de temps, jusqu'à 30 secondes. En utilisant les trois méthodes de manière couplée, le temps de calculs diminue pour arriver à seulement 20 secondes. Une utilisation couplée des méthodes est bien plus efficace car permet d'éliminer immédiatement toutes les possibilités évidentes, indétectables en *backtracking*, pour ensuite l'utiliser éventuellement si la grille n'est pas résolue.

2. Les résultats peuvent varier d'un machine à l'autre comme nous avons pu le constater, il est donc surtout intéressant de noter les différences d'ordres de grandeurs entre les différents résultats.

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.

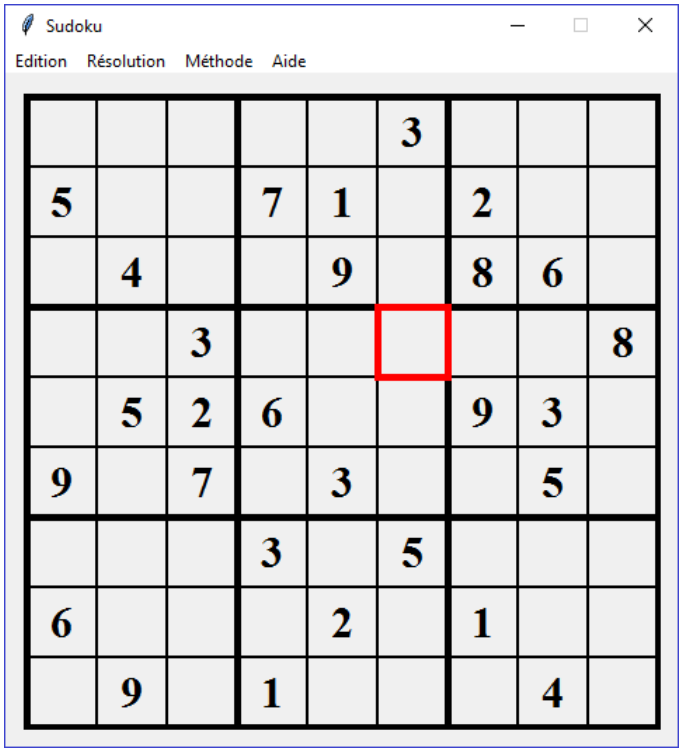

FIGURE 4.11 – Interface graphique du Sudoku

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.

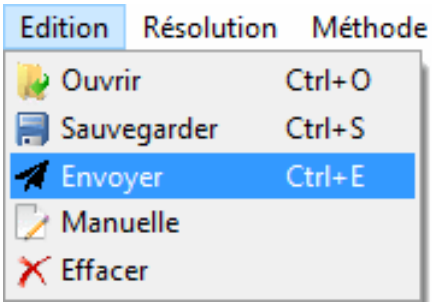

FIGURE 4.12 – Menu *Édition* de l'interface graphique

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.

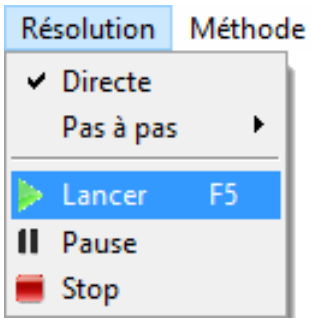

FIGURE 4.13 – Menu *Résolution* de l'interface graphique

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

4.5 Contrôle des moteurs et écriture

Moteurs pas-à-pas

Il nous apparaissait nécessaire de développer une programmation permettant l'exécution simultanée de plusieurs instructions, notamment pour les moteurs. En effet, les moteurs doivent absolument fonctionner de manière synchrone afin d'obtenir des déplacements les plus précis possibles. C'est pour cela que nous nous sommes basés sur ce qui pourrait être qualifiée de "programmation parallèle" qui permet l'exécution de plusieurs processus simultanément. Cela est rendu possible en Python par l'utilisation de la bibliothèque *threading* depuis laquelle nous utilisons *Thread* (signifiant *processus* en français). Pour piloter les moteurs, il suffit d'envoyer des impulsions dans les bobines pour faire tourner les moteurs dans le bon sens. La fréquence d'envoi de ces impulsions conditionne la vitesse des moteurs. Les impulsions doivent être envoyés au maximum toute les 10 ms, ce qui constitue la vitesse maximum des moteurs. Ces impulsions sont envoyés au deux moteurs de manière cycliques, un cycle étant constitué de quatre séquences [3]. Le script d'écriture des digits transmet à celui des moteurs une liste des coordonnées des points à parcourir. Ces coordonnées sont ensuite converties en déplacements élémentaires pour les deux moteurs, ce qui se traduit alors en impulsions et permet les déplacements des moteurs.

3. cf section 3.1 à la page 9

Coordonnées polaires

Par soucis de compacité et ainsi s'inscrire dans l'optimalité du programme, nous avons décidé de développer une structure se basant sur les coordonnées polaires, permettant des dimensions maximum de 40 par 10 cm. L'utilisation de telles coordonnées nécessitait une plus grande précision et relevait donc d'un plus grand défi technique. Les coordonnées polaires ne sont de plus, que très peu utilisés pour ce type de système, ce qui nous à incité à développer une structure se basant sur ces coordonnées.

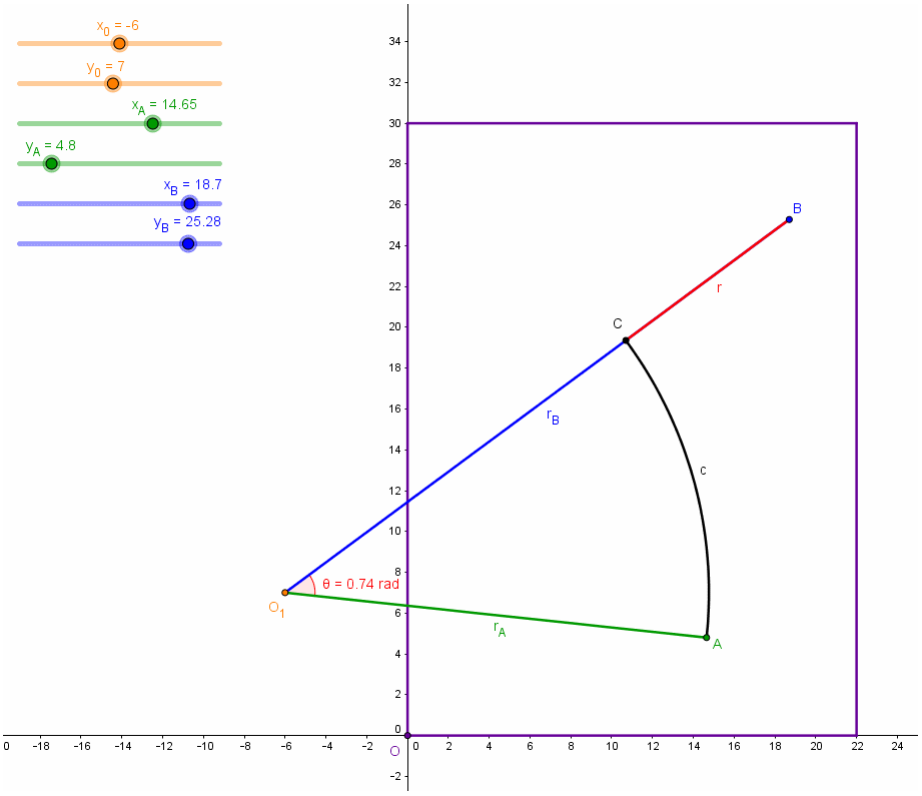

FIGURE 4.14 – Changement de repères réalisé avec *Geogebra*

Sur la figure 4.14, le point O_1 correspond au centre de la roue dentée. Le point A correspond à la position du bras et le point B correspond à la position que l'on souhaite atteindre. On peut noter que le bras doit pivoter d'un angle $\theta = 0.74$ rad et doit se déplacer d'une distance algébrique r pour atteindre ce point B. Ces deux données sont transmises aux moteurs qui vont se déplacer d'un nombre de pas équivalent pour l'un à θ et pour l'autre à r. On comprend alors qu'il faut suffisamment de points pour éviter de tracer des lignes courbes, en effet si A et B se situent à la même distance du centre O_1, le stylo tracera un arc de cercle Pour avoir accès à θ et r, il faut convertir les coordonnées des point A et B en coordonnées polaires de centre O_1. On notera x_A et y_A les coordonnées cartésiennes du point A et r_A et θ_A ses coordonnées polaires de centre O_1. Les coordonnées du point O_1 seront eux indicées par 0. On a alors :

$$\begin{cases} x_A = r_A \cdot \cos\theta + x_0 \\ y_A = r_A \cdot \cos\theta + y_0 \end{cases} \quad \text{et} \quad \begin{cases} r_A = \sqrt{(x_A - x_0)^2 + (y_A - y_0)^2} \\ \theta_A = \frac{y_A - y_0}{|y_A - y_0|} \cdot \arccos\left(\frac{y_A - y_0}{r_A}\right) \end{cases}$$

Comme $r_A = 0 \iff A = O_1$ et que A est un point de la feuille alors que O_1 ne l'est pas, r_A ne peut être nul [4]. On en déduit que θ_A est bien définie quelque soit A. En notant nb_{step1} (resp. nb_{step2}) le nombre de déplacements élémentaires du moteur 1 (resp. du moteur 2), on obtient :

$$\begin{cases} r = r_B - r_A \\ \theta = \theta_A - \theta_B \end{cases} \text{ et } \begin{cases} nb_{step1} = E(\frac{r}{r_{step}} + \frac{1}{2}) \\ nb_{step2} = E(\frac{\theta}{\theta_{step}} + \frac{1}{2}) \end{cases}$$

Les entiers relatifs nb_{step1} et nb_{step2} sont les deux informations véritablement importantes pour les moteurs et qui correspondent aux nombre de séquences de leurs déplacements.

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 4.15 – Chiffres d'une précision théorique assez faible

Le script permettant de tracer les chiffres prend deux paramètres, à savoir la hauteur des chiffres et la précision attendue. Lors des tests du bras mécanique, ces deux paramètres se sont révélés déterminants pour obtenir le meilleur résultat possible. Les chiffres en tant que tel sont constitués de lignes et d'arc de cercle permettant d'obtenir un résultats assez satisfaisants.

En augmentant la précision théorique, on obtient des chiffres quasiment continus permettant d'obtenir la figure 4.16.

Il faut prévoir dans le script la possibilité d'envoyer une impulsion au servomoteur afin qu'il lève le stylo lorsqu'un chiffre a été tracé afin d'éviter que les chiffres ne soient reliés entre eux, comme le suggère la figure 4.17.

4. De plus, du fait du mécanisme, le point O_1 ne peut pas être atteint par le stylo

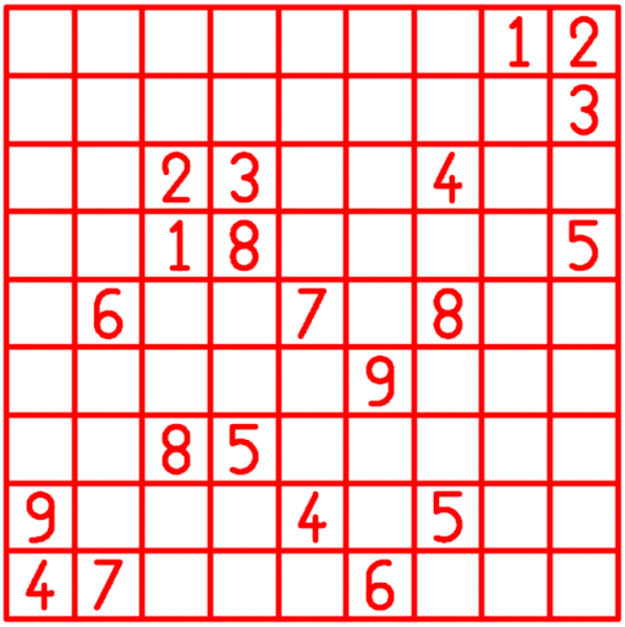

FIGURE 4.16 – Grille de sudoku en point par point tracée avec *MatplotLib*

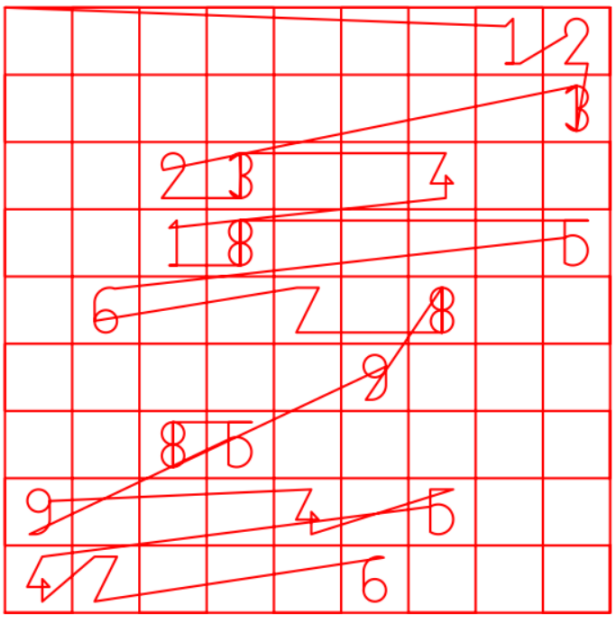

FIGURE 4.17 – Grille de sudoku tracée avec *MatplotLib*

Connexion entre la Raspberry Pi et un ordinateur distant

Une Raspberry Pi peut être utilisée comme un véritable ordinateur, il suffit pour cela de lui brancher un écran et un clavier. Le problème de cette utilisation réside dans l'utilisation de l'écran, en effet la *Raspberry Pi* dispose d'une sortie HDMI sur laquelle se branche l'écran. Cependant l'utilisation de cette sortie sur notre bras n'aurait pas du tout été pratique et nous n'avions pas forcément un écran à disposition. C'est pour cela que nous avons développé la possibilité de contrôler la *Raspberry Pi* via une connection sécurisée, le *SSH* (*Secure Shell*). Cependant ce type de connection ne permet pas l'échange d'in-

formations directes entre les deux appareils mais permet de prendre la main à distance sur la *Raspberry* (permettant par exemple d'exécuter des scripts à distance).

C'est pour cela que nous avons utilisé un module python appelé *socket* qui permet l'échange d'informations entre un serveur et un client. Lors de la mise sous tension de la *Raspberry Pi*, plusieurs de nos scripts, à savoir le script principal et le script qui démarre le serveur, sont automatiquement exécutés. Sur le PC distant, il suffit de démarrer l'interface graphique, qui est alors un client pouvant se connecter au serveur de la *Raspberry*. Dans le menu édition, le bouton *Envoyer*[5] permet de transmettre la grille courante à la *Raspberry* qui va se charger de l'écrire physiquement. L'écran LCD sert alors à vérifier que les informations ont bien été transmises.

5. Cf Figure 4.12 à la page 21

Conclusion

Ce TIPE a été très bénéfique et nous a tout d'abord permis d'améliorer les compétences acquises l'an dernier en Python.

De plus, de nombreuses nouvelles compétences ont du être mises en œuvre :

- le contrôle de moteurs à l'aide d'une Raspberry Pi
- la conception de pièces en 3D

Tous les scripts et ce dossier sont disponibles sur GitHub [6]

6. Sudoku Plotter - `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.