

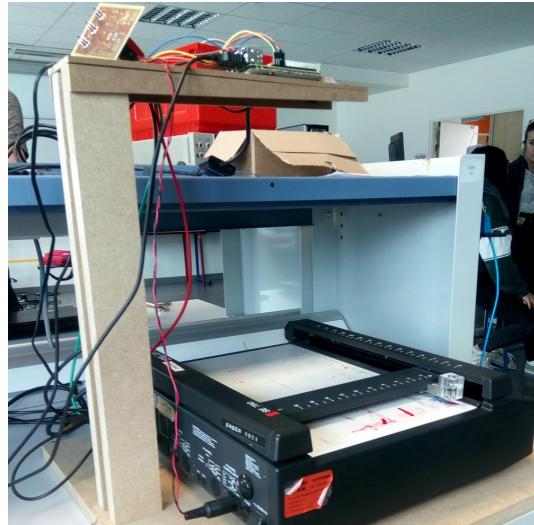


Filière Signal, Image, COmmunication et Multimédia

Projet collectif de 2^{nde} année - 2017/2018

Rapport Final

Un robot solveur de sudoku



Auteurs :

Zaineb AMOR
Valentin BONNET
Simon SUBIAS
Sébastien TAURAND

Tuteur :
Pr. Bertrand RIVET

11 mai 2018

Remerciements

Nous tenions par le présent rapport à remercier nos encadrants M. Vincent FRISTOT et Mathias VOISIN-FRADIN pour l'aide et les conseils prodigués tout au long de ce projet. Nous remercions également notre tuteur M. Bertrand RIVET pour nous avoir conseillé sur la marche à suivre du projet.

Merci aussi à l'ensemble du personnel technique de Phelma pour nous avoir apporté leur aide pour la réalisation de nombreux éléments de notre montage tels que le circuit imprimé de commande ou encore la potence de support pour la caméra : M. Julien TRAVEAUX, Mme Florence NOËL, Mme Sophie CORNU, M. Jean-Paul SAUNIER et M. Gilles LUCIANI.

Table des matières

1 Contexte	4
1.1 Cahier des charges	4
1.2 Organisation de l'équipe	5
1.3 Outils et méthodologie de travail	6
1.3.1 Plateforme physique	6
1.3.2 Plateforme logicielle	6
1.3.3 Méthodologie	6
2 Rapport technique	7
2.1 Partie logicielle	7
2.1.1 Calibration	7
2.1.2 Traitements préliminaires de l'image	9
2.1.3 Détection de la grille	10
2.1.4 Identification des digits	12
2.1.5 Résolution de la grille	15
2.2 Partie matérielle	17
2.2.1 Support et carte PCB	17
2.2.2 Prise en main de la table traçante	17
2.2.3 Algorithme d'écriture	20
2.2.4 Gestion des contraintes	21
2.2.5 Remplissage du Sudoku	23
3 Bilan du travail	26
3.1 Réalisation finale	26
3.2 Évaluation du coût de fabrication	28

Chapitre 1

Contexte

1.1 Cahier des charges

L'objectif de ce projet est de réaliser un dispositif permettant de lire, résoudre et remplir une grille de sudoku. La lecture se fera avec une caméra et le remplissage avec une table traçante analogique donnée (Servogor 790[11]).

Les attentes des clients sont les suivantes :

- Le système doit être simple d'utilisation et autonome. Idéalement, il suffirait de poser la grille de sudoku à traiter sur la table traçante et d'appuyer sur un bouton ou une touche de clavier pour démarrer le dispositif.
- La résolution ainsi que l'écriture du sudoku doit pouvoir être faite quelle que soit l'orientation du sudoku sur la table traçante.
- La taille de la grille à résoudre n'est pas fixée, mais doit être toutefois compris entre 6x6 cm et 16x16 cm du fait des limitations de la table.
- Les chiffres de la grille doivent être sous forme de caractères imprimés, la police d'écriture étant libre.
- Le traitement ne doit pas être trop long.

1.2 Organisation de l'équipe

La première tâche a été d'identifier l'ensemble des tâches à effectuer, à les répartir au sein de l'équipe et à les planifier.

Tâche	Membres impliqués
Pré-traitement de l'image	Sébastien et Zaineb
Extraction de la grille	Sébastien et Zaineb
Résolution de la grille	Valentin
Reconnaissance des digits	Valentin
Contrôle de la table traçante	Simon et Sébastien
Écriture de la grille	Simon
Conception du support et du PCB	Zaineb et Valentin

TABLE 1.1 – Répartition des tâches

La première étape consiste en la prise en main du matériel : table traçante (fonctionnement, plages d'utilisation, caractérisation...) et Raspberry Pi (paramétrage, SSH, mise au point de la caméra, utilisation des ports GPIO...). Une fois le Raspberry Pi maîtrisé, le développement logiciel a démarré, autant sur le traitement d'image que sur l'écriture des digits. Une fois le livrable 1 terminé, le début de la chaîne de la capture d'image à l'extraction de la grille sous forme d'une matrice est fonctionnel. S'attaque alors la partie "matérielle" du projet qui concerne l'écriture des chiffres sur la grille. En complément, la construction du dispositif a été menée avec la conception du plateau en bois et d'une PCB de contrôle.

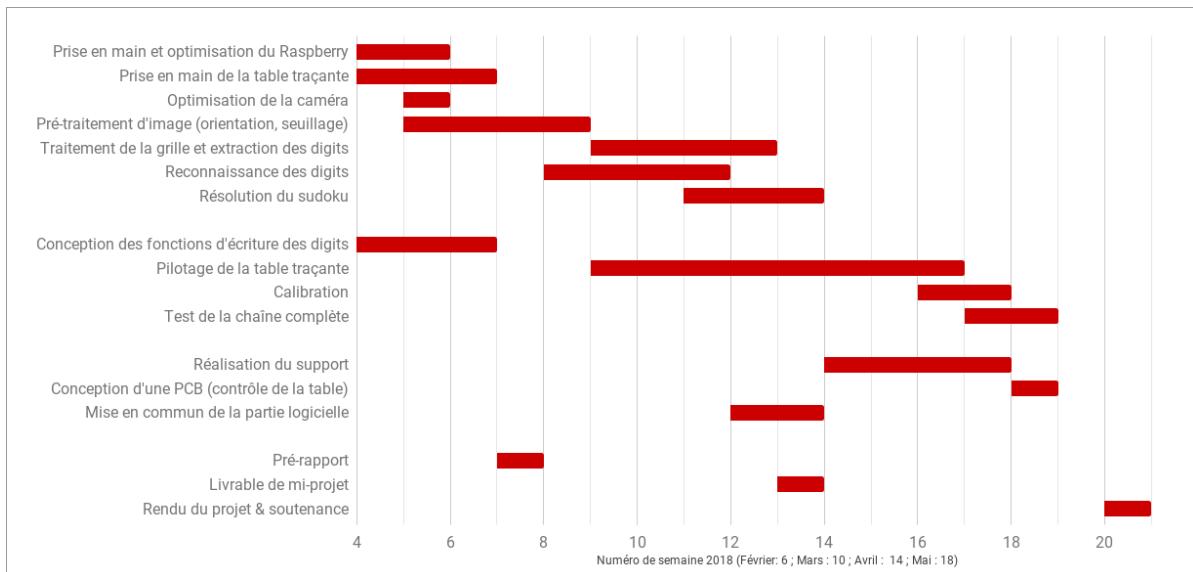


FIGURE 1.1 – Diagramme de Gantt réel

1.3 Outils et méthodologie de travail

1.3.1 Plateforme physique

Nous avons retenu le Raspberry Pi 3B comme plateforme matérielle compacte et à bas coût, un Arduino n'étant absolument pas adapté à l'acquisition et au traitement d'image.

Les sorties analogiques servant à piloter la table traçante sont basées sur les deux sorties PWM hardware du Raspberry Pi. Nous avons bien pris garde de choisir une implémentation PWM hardware[10][3] et non logicielle car cette dernière est beaucoup moins précise et très sensible au taux d'utilisation du processeur.

1.3.2 Plateforme logicielle

Le développement du projet a été effectué sous Rasbian Stretch en Python 3. Nous nous sommes basés sur la bibliothèque OpenCV 3.4.1[7] pour tout ce qui a trait au traitement d'images[5] et à l'apprentissage statistique[6]. L'acquisition des images de la caméra utilise le module PiCamera. Numpy est aussi utilisé de façon extensive.

1.3.3 Méthodologie

Nous avons travaillé en suivant les préceptes de la méthode Agile. Nous avons implanté les fonctions principales de façon modulaire et incrémentale, en testant de façon régulière l'enrichissement du code. Une fois les modules suffisamment matures, nous les avons intégrés ensemble et testés.

Nous avons parfois dû revoir les définitions des fonctions et structures de données associées afin d'optimiser l'interface entre les modules.

Nous avons aussi été amenés à changer des algorithmes que nous avions imaginés en début de projet mais qui se sont avérés insuffisamment performants.

Chapitre 2

Rapport technique

2.1 Partie logicielle

La partie logicielle concerne la première partie de la chaîne, du traitement d'image à la résolution de la grille en passant par la détection de grille et l'identification des chiffres.

2.1.1 Calibration

Pour pouvoir tracer dans les cases de la grille, il est nécessaire de disposer d'une bijection entre les coordonnées [PWMx, PWMy] et [CAMERAx,CAMERAy]. A cause des différentes imperfections géométriques de la table traçante, du support de la caméra et de la caméra en elle-même, il existe une distorsion de perspective entre les deux systèmes de coordonnées.

OpenCV dispose de fonctions pour gérer ces corrections de perspective :

- `getPerspectiveTransform` permet de déterminer la matrice de transformation de perspective à partir de quatre points de référence
- `perspectiveTransform` permet de changer de système de coordonnées pour des points
- `warpPerspective` permet de corriger la perspective d'une partie d'image

Une calibration est nécessaire afin de pouvoir établir cette bijection. Elle se fait à chaque mise en route du Raspberry Pi. On effectue le tracé du rectangle d'aire maximale utile, puis on prend le prendre en photo et on détermine ses sommets. A partir de là, on dispose donc des quatre points permettant de faire la bijection position physique / position sur image. La compensation est aussi efficace pour corriger les erreurs de facteur d'échelle des sorties PWM dues aux imprécisions de la tension d'alimentation du Raspberry Pi (variation inférieure à 5% de la pleine échelle). La matrice de perspective est stockée sur le disque pour

être utilisée entre deux calibrations.

Le programme de calibration prend en entrée deux images : celle de la table traçante avant traçage du rectangle et celle après traçage du rectangle, effectue après une soustraction entre ces deux images pour n'avoir que le rectangle dans lequel nous pouvons écrire et soustrait la moyenne de l'image de celle-ci pour éviter un quelconque défaut d'éclairage. La fonction utilisée est `goodFeaturesToTrack` de la librairie **OpenCV**. Cette fonction retournera tous les coins détectés sur l'image. Mais cette fonction peut détecter des points (qui ne sont pas les points extrêmes du rectangle) appartenant au rectangle comme étant des coins, un tri des coins trouvés est alors effectué. Pour cela les coins à gauche et les coins à droite sont séparés. Une fois ces deux groupes formés, il faut trouver les points extrêmes, c'est-a-dire les coins présentant les coordonnées maximums et minimums sur x et sur y . De ce qui a pu être remarqué, ces points extrêmes détectés jusqu'à maintenant, peuvent être en réalité de faux coins un peu en dehors du rectangle. Pour corriger cela la moyenne sur y des points trouvés est calculées. Les points appartenant vraiment au rectangle seront alors ceux dont la coordonnée sur y est égale à la moyenne calculées plus ou moins un delta. C'est ainsi que nous arrivons à n'avoir que les points appartenant aux deux côtés du rectangle. La dernière étape est alors de prendre les points ayant les coordonnées maximums et minimums sur x cette fois.

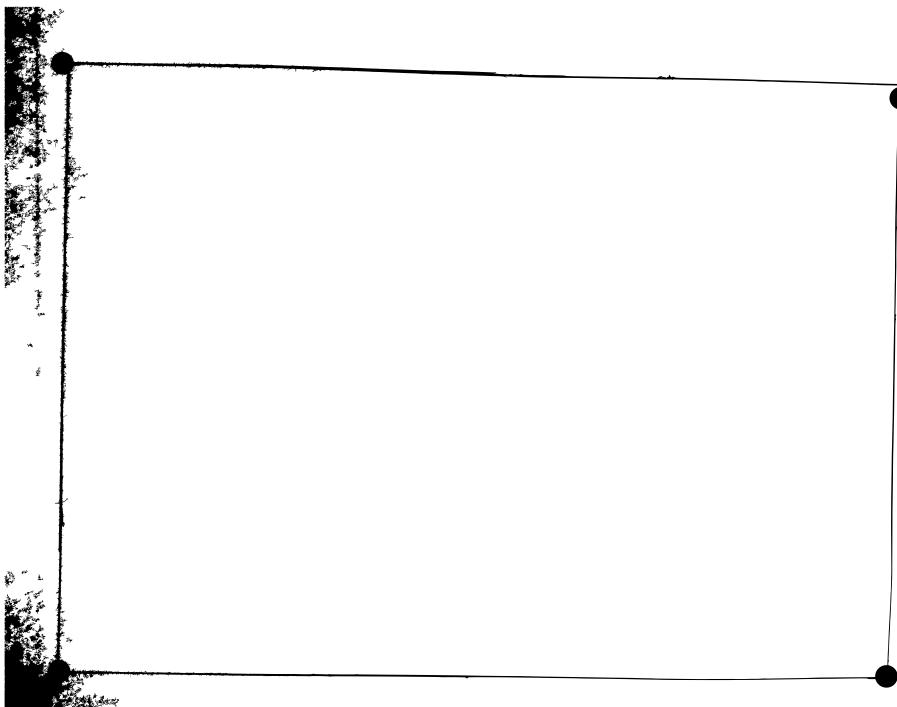


FIGURE 2.1 – Coins détectés sur le rectangle

2.1.2 Traitements préliminaires de l'image

Image source

L'image est capturée par la PiCamera (8M pixels couleur) et est convertie en 2M pixels à 256 niveaux de gris. Cette réduction de taille est effectuée pour ne pas imposer une charge de calcul trop importante dans les étapes suivantes du traitement tout en laissant une résolution suffisante pour lire et identifier les grilles de sudoku et leur contenu.

Afin de faciliter les traitements de seuillage et d'extraction de contour, il est impératif que la mise au point de la caméra ait bien été réglée pour la distance entre la caméra et la surface de la table traçante.

Seuillage de l'image

Afin de rendre possible les opérations ultérieures sur l'image, il est nécessaire de passer l'image en noir et blanc.

Les méthodes suivantes ont été testées :

- Seuillage de Otsu (recommandé pour les images à deux pics de luminance dans l'histogramme)
- Méthode de Canny[2]
- Seuillage adaptatif gaussien

Le seuillage de Otsu s'est avéré peu performant sur les gradients d'éclairage de l'image ainsi que perturbé par la présence d'éléments externes à la grille de sudoku.

Le seuillage adaptatif gaussien s'est avéré le plus performant dans le seuillage des grilles de sudoku en elles même et de leur contenu. On remarquera toutefois que le niveau de bruit en dehors des grilles de sudoku est significativement augmenté par cette méthode, même si ce n'est pas pénalisant pour la suite de l'algorithme de reconnaissance. Le filtre gaussien s'avère le plus efficace lorsque la taille de son masque est proche de la taille d'une case du sudoku. Ce choix permet en particulier d'obtenir des lignes de grille non interrompues, ce qui est essentiel pour la reconnaissance du contour de grille.

Extraction de contours

La méthode d'extraction des lignes de grille par transformée de Hough, qu'elle soit globale ou statistique a été implémentée et testée mais non retenue car plus sensible aux éléments externes à la grille. Un autre défaut de la méthode était la difficulté à identifier les lignes proches des bords de l'image ainsi qu'une mauvaise tolérance aux distorsions

géométriques de l'image (segments de droite courbés).

Afin de rendre l'algorithme de reconnaissance des grilles robuste aux éléments parasites de l'image, une extraction de contour hiérarchique est préférée.

2.1.3 Détection de la grille

La détection de la grille est effectuée par extraction de contours hiérarchisés (pour chaque contour, on dispose en particulier d'une information sur le parent (contour englobant ce contour) et la présence d'enfants (contours englobés)).

Ne seront considérés que certains contours ayant des propriétés spécifiques à une grille de sudoku :

- Le contour externe de la grille a une taille significative dans l'image
- Une grille contient 81 cases (contours enfants) approximativement carrées et de taille connue par rapport à la taille de la grille externe
- Il peut aussi contenir d'autres contours mais de tailles beaucoup plus petites que les cases qui seront éliminés du traitement
- Les cases peuvent être vides ou contenir un ou plusieurs contours
 - Le plus grand contour contenu, s'il a une taille suffisante par rapport à la taille de la case est considéré comme un chiffre
 - Les autres contours sont éliminés

Pour que cet algorithme fonctionne, il est nécessaire que les contours soient fermés, le seuillage de l'image est ainsi un point critique du traitement.

Pour chaque grille identifiée, les contours des cases sont approximés par un quadrilatère dont on récupère les coordonnées pour servir de référence à l'opération ultérieure d'écriture.

Quelques images représentant le processus de traitement d'image et les résultats intermédiaires obtenus sont représentées figure 2.2 à 2.6.

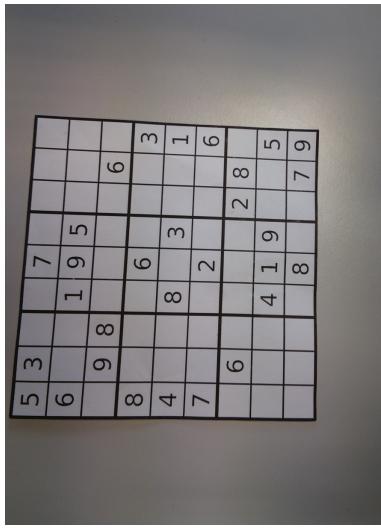


FIGURE 2.2 – Grille en 256 niveaux de gris

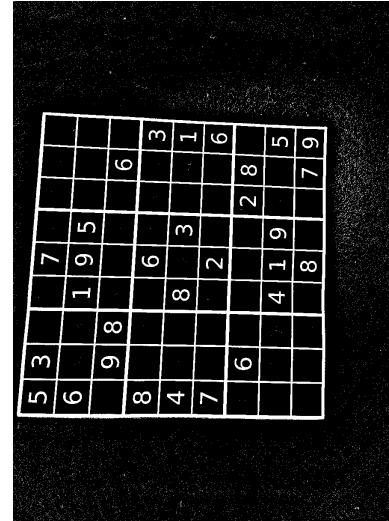


FIGURE 2.3 – Image seuillée

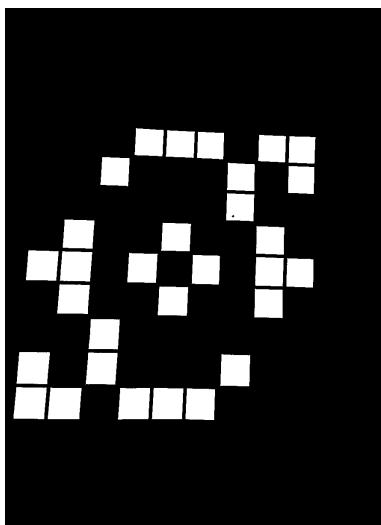


FIGURE 2.4 – Masque pour récupération des cases pleines et retrait des impuretés

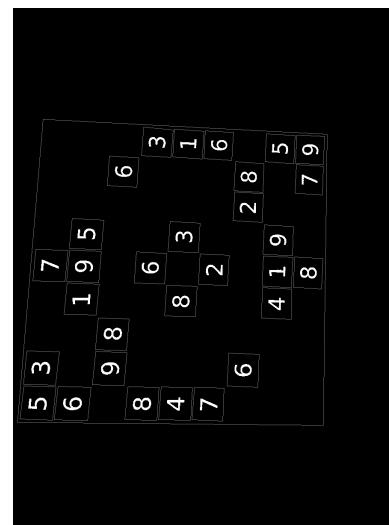


FIGURE 2.5 – Chiffres et quadrilatères de la grille et des cases

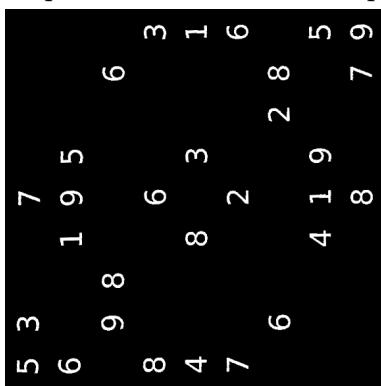


FIGURE 2.6 – Image finale permettant de récupérer les digits

2.1.4 Identification des digits

La reconnaissance des chiffres est réalisée via l'algorithme de classification des *k plus proches voisins*, implanté dans la librairie **OpenCV**[6]. Pour l'apprentissage, nous utilisons la base de donnée **Char74k EnglishFnt.tgz**[1] comportant 1016 caractères imprimés pour chaque digit.

La méthode des *k plus proches voisins* exploite les caractéristiques des images de la base de donnée pour situer ces dernières dans un espace multidimensionnel, et leur assigner une classe (ici, la valeur du digit) : c'est la phase d'apprentissage.

Lors de la phase de reconnaissance, l'image à reconnaître formera un nouvel individu, et sera située dans cet espace. Des calculs de distance Euclidienne permettent de trouver les plus proches “voisins” de cet élément, et ainsi de récupérer leur classe. Une moyenne des classes obtenues permettra d'assigner à l'image sa valeur.

Lors de la phase d'apprentissage, un ratio apprentissage/test est demandé. Il permet de séparer la base de donnée d'images en deux : une partie pour l'apprentissage, et l'autre pour évaluer la précision de l'apprentissage à partir d'un jeu de données labellisé. Les images utilisées pour l'apprentissage sont sélectionnées de manière aléatoire.

L'algorithme est capable, à la demande de l'utilisateur, de renvoyer un nombre de plus proches voisins *k* permettant une précision optimale de reconnaissance, en se basant sur la phase de test. Cette valeur de *k* est toutefois renvoyée à caractère indicatif, et doit, pour le moment, être manuellement entrée dans l'algorithme de reconnaissance. En effet, dans le système final, la fonction d'apprentissage ne sera pas exécutée. Seront alors uniquement stockés en mémoire les fichiers **.data** contenant les données apprises.

La figure 2.7 montre que la précision de l'algorithme augmente avec le ratio apprentissage/test, ce qui est logique. Toutefois, un apprentissage lourd génère une base de donnée conséquente qui risque de ralentir l'algorithme lors de la reconnaissance (entraînement du modèle). Inutile donc de choisir un ratio trop élevé, un ratio de 0.05¹ donne déjà une précision moyenne avoisinant les 88%.

Le maximum observé sur ces courbes pour un seul voisin considéré traduit le fait que la classe du plus proche voisin correspond la plupart du temps à la classe du digit. Il est important de noter que la phase de test n'est réalisée qu'avec des images de la base de données, i.e. des images idéales. Dans le cas réel où la forme des digits aura été altérée par la caméra ainsi que par les pré-traitements successifs, nous obtenons de meilleurs résultats avec un nombre de plus proches voisins relativement élevé (correspondant au second maxi-

1. correspondant à l'apprentissage de 52 images par digits

FIGURE 2.7 – Précision de l'algorithme selon le nombre de plus proches voisins considérés, pour plusieurs valeurs de ratio apprentissage / test

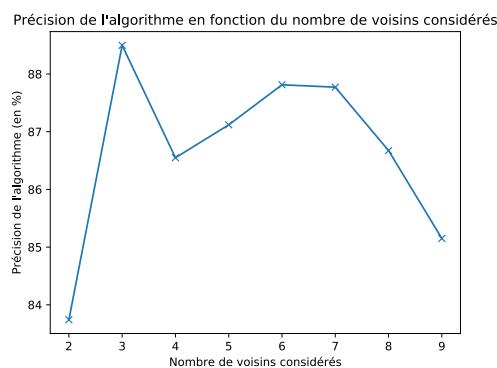


FIGURE 2.8 – ratio=0.05

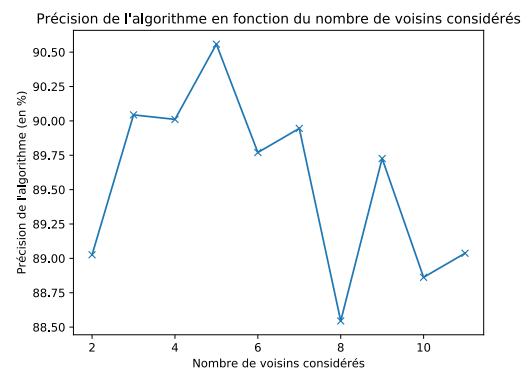


FIGURE 2.9 – ratio=0.1

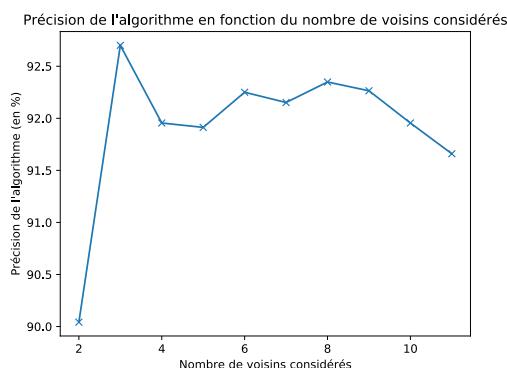


FIGURE 2.10 – ratio=0.3

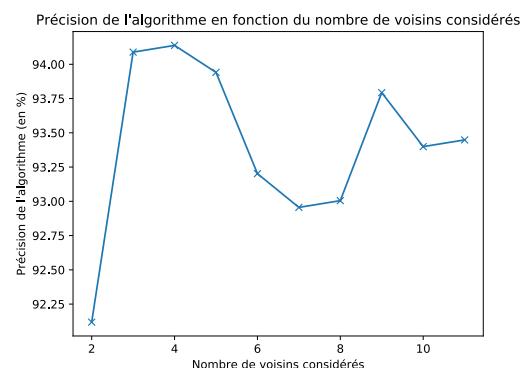


FIGURE 2.11 – ratio=0.8

mum observé sur les courbes, soit environ 8).

Remarque : la base de donnée comprend des digits arborant des polices "fantaisistes". Ils permettent de tester en profondeur la précision de l'algorithme, et expliquent le fait que nous n'obtenons pas une précision plus proche des 100% à chaque étape apprentissage/test malgré un ratio relativement élevé. En pratique, les polices des sudoku sont conventionnelles, comme la plupart des polices de la base de donnée, ce qui conviendra très bien à notre utilisation.

Pour la reconnaissance, comme précisé précédemment, l'algorithme est capable de déterminer l'orientation du sudoku redressé (0° , 90° , 180° ou 270°). Pour cela, quatre reconnaissances selon ces 4 angles sont réalisées, puis celle présentant la distance moyenne entre plus proches voisins la plus courte est choisie.

La reconnaissance s'opère case par case, chaque case étant isolée par une fonction basique de segmentation qui divise en cases de dimensions identiques l'image représentant le sudoku (Cf. Figure 2.12). Pour accélérer la reconnaissance, et ainsi éviter de devoir réaliser quatre reconnaissances complètes, la validité de la grille de sortie est testée après chaque reconnaissance de digit. Ainsi, si l'algorithme détecte deux chiffres '8' qui sont potentiellement situés dans la même ligne, colonne ou bloc, la reconnaissance pour l'angle donné s'arrête immédiatement.

Afin d'assurer une sécurité supplémentaire, certains critères sont ajoutés pour restreindre les dimensions du contour rectangulaire autour des digits (cf. image suivante). Si ces derniers sont trop fins, trop petits, ou trop excentrés de la case, la reconnaissance ne s'opère pas et la case est considérée comme vide.

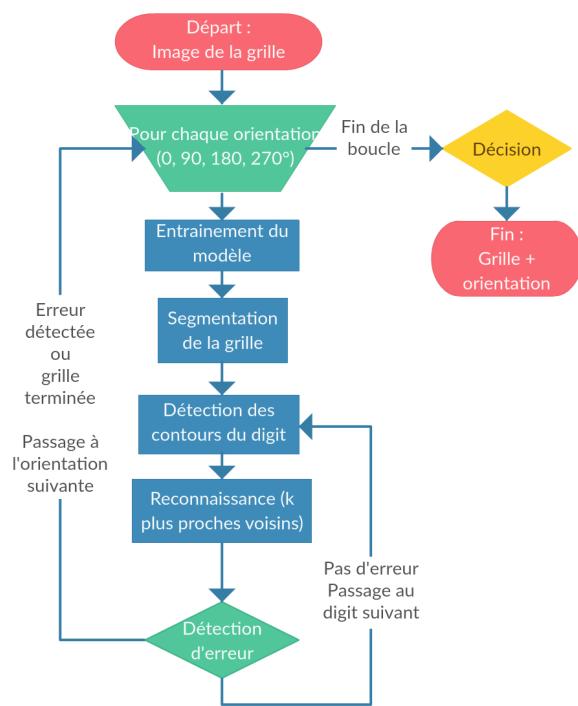


FIGURE 2.12 – Algogramme pour la reconnaissance de digits



FIGURE 2.13 – Exemples de digits ciblés par la détection de contours.

Les valeurs renvoyées à partir des images mal orientées seront probablement fausses et généreront une grille non valide. L’invalidité de la grille permettra de conclure sur l’invalidité de l’orientation testée.

2.1.5 Résolution de la grille

Pour la résolution du sudoku, nous utilisons une méthode dite de “backtracking”[9]. Il s’agit d’une méthode relativement simple de résolution par “brute-force” : toutes les combinaisons possibles sont essayées pour chaque case vide. Dès que le digit entré ne respecte plus les règles du sudoku, l’algorithme retourne en arrière et change le digit précédemment entré.

Pour réaliser un tel retour en arrière, le cœur de l’algorithme est une fonction récursive, `aux(i,j)` modifiant la grille et retournant un booléen :

- `True` lorsque la grille est résolue, i.e. par convention lorsque le coefficient (9,0) est appelé (la limite de la grille étant (8,8) pour un sudoku 9x9).
- `False` lorsque le coefficient rentré en (i,j) ne respecte pas les conditions de la grille (i.e. est déjà présent dans la ligne i, colonne j ou dans le bloc).
- `aux(case_suivante(i,j))` pour passer à la case suivante vide lorsque le digit rentré en (i,j) est valide.

Un algorithme de backtracking ayant généralement une complexité en $O(e^n)$ [8], il est nécessaire d’optimiser son déroulement afin d’avoir un temps de calcul acceptable. Pour cela, la fonction `case_suivante(i,j)` qui détermine les coordonnées de la prochaine case à remplir, est optimisée. Le but est de remplir en priorité les cases ayant le moins de possibilités, ce qui maximise la probabilité d’y entrer le chiffre correct. Cette optimisation est assez intuitive et proche du comportement adopté par l’homme pour résoudre une grille.

En pratique, cette optimisation conduit à calculer la *matrice des possibilités* à chaque fois qu’un nouveau chiffre est entré dans la grille. Le calcul de cette matrice, réalisé par la fonction `ordre_backtracking` est de complexité $O(n^3)$, ce qui reste acceptable dans le cas $n = 9$. La fonction `case_suivante` permettant de déterminer les coordonnées de la prochaine case à remplir sélectionnera la case affichant un nombre de possibilités minimal.

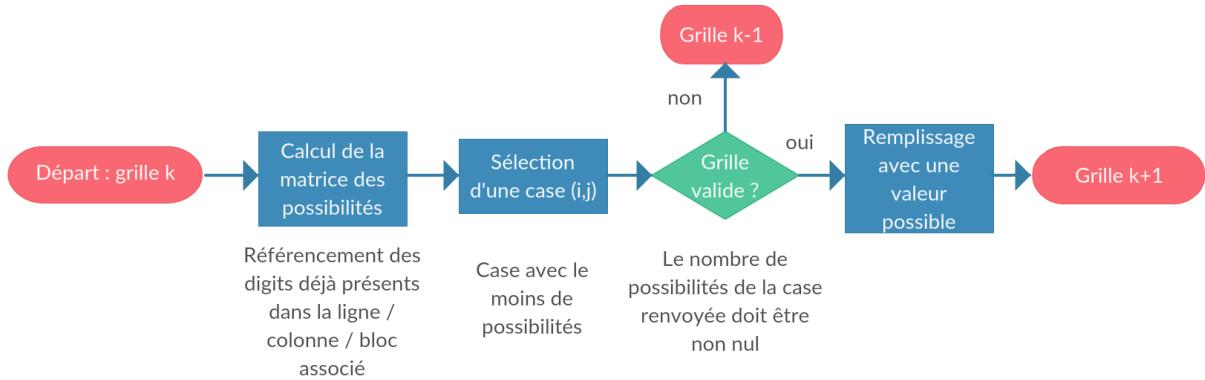


FIGURE 2.14 – Descriptif d'une étape de l'algorithme de résolution. Réalisé via *Createely*

Sudoku									Matrice des possibilités									
5	2	9		0	7	0		0	6	0	0	0	3	0	3	3	0	2
3	0	4		0	0	5		1	0	0	0	1	3	4	0	0	4	3
1	8	6		0	0	0		0	0	5	0	0	2	4	3	5	5	0
0	0	0		0	0	0		0	0	2	0	0	0	2	4	3	5	5
0	0	2		0	0	0		0	0	6	0	0	1	3	4	0	0	4
7	0	0		0	0	8		0	5	0	0	0	0	0	0	0	0	0
6	0	0		2	0	0		0	0	0	0	0	0	0	0	0	0	0
0	0	0		0	0	9		0	0	3	0	0	0	0	0	0	0	0
2	0	7		4	0	0		0	0	0	0	0	0	0	0	0	0	0

TABLE 2.1 – Exemple de sudoku et de matrice des possibilités associée

Les chiffres nuls dans la grille correspondent aux valeurs à remplir ; il n'y a bien qu'une seule possibilité pour le chiffre en rouge (7). S'agissant de première valeur minimale de la matrice des possibilités, cette case sera traitée lors du prochain appel récursif.

Au final, l'algorithme met² quelques millisecondes à quelques secondes pour résoudre un sudoku 9x9 selon son niveau de difficulté.

2. Les temps indiqués sont liés à une exécution sur PC. Pour le Raspberry Pi, il faut compter 2 à 3 fois plus de temps.

2.2 Partie matérielle

2.2.1 Support et carte PCB

Notre système matériel étant nécessairement fixe (la table traçante n'est pas censée changer, ni le système de capture d'image), un support devait nécessairement être construit pour fixer la position de la caméra vis à vis de la table traçante. Après conseil, une solution basée sur l'utilisation d'un serre-joint paraissait délicate, à la fois pour la fabrication et pour une fixation durable. Par conséquent, le support construit, représenté figure 2.16, est constitué de planches de bois assemblées. La mise au point de la caméra a été manuellement réglée par nos soins pour une image nette à une distance avoisinant les 40 cm de la surface de la table traçante. Cette hauteur est également nécessaire pour couvrir la totalité de la surface de la table traçante. Afin de fixer la position de la table par rapport à la caméra, des trous ont été réalisé pour marquer dans le bois la position des patins de la table. La caméra et le Raspberry sont fixés de manière temporelle (scotch, carton et colle) pour des raisons de réutilisation de matériel. Afin d'éviter que la planche soutenant la caméra n'oscille, des arrêtes en bois ont également été fixées sur le côté.

Suite à quelques problèmes de contacts électroniques, concevoir une carte PCB était primordial. Étant donné que nous n'utilisons qu'une partie restreinte des ports GPIO, les connecteurs présents sur la PCB ne correspondent volontairement pas au nombre de ports GPIO du Raspberry Pi 3B. Des trompeurs (connecteurs de taille différente) sont mis en place pour repérer chaque branchement. La PCB présente une LED permettant de signaler une erreur³ dans l'exécution de l'algorithme.

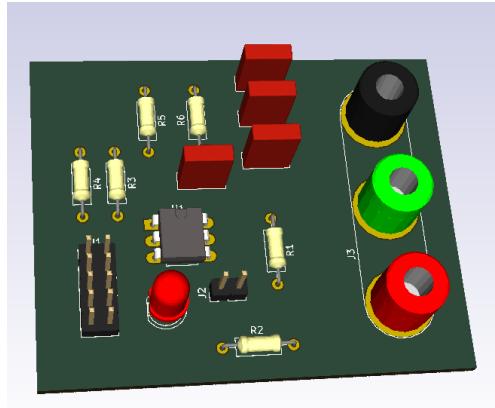


FIGURE 2.15 – Visualisation 3D de la PCB

2.2.2 Prise en main de la table traçante

Une fois la grille résolue de manière numérique le système doit être capable de remplir la grille originale de manière très précise. Pour cela la table traçante s'est révélée être

3. Erreur pouvant être identifiée par le nombre de clignotements de cette dernière

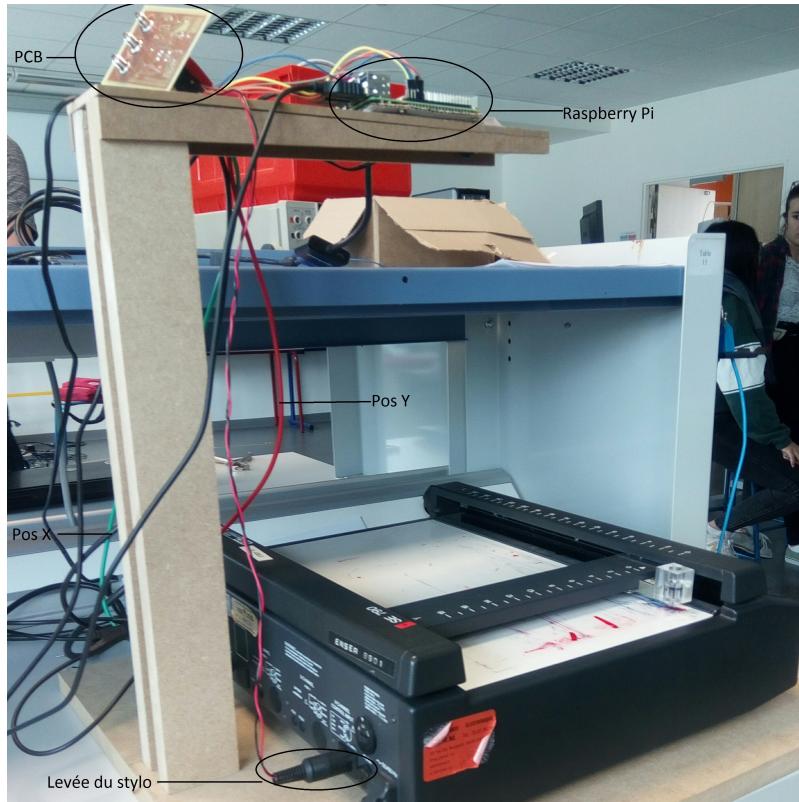


FIGURE 2.16 – Système matériel

l'outil adapté. Elle permet de déplacer, avec une précision plus fine que le millimètre, un stylo sur un axe horizontal x et sur un axe vertical y et de le lever où le baisser. Les positions en cm sur x et y sont images de tensions envoyées sur deux bornes : la position est égale à la tension divisée par le calibre. Pour baisser le stylo il suffit de fermer son circuit.

Afin de contrôler ces trois paramètres, le Raspberry Pi s'est révélé très pertinent. Sur X et Y il faut envoyer une tension qui évolue dans le temps, ce que ne permet pas le Raspberry. Mais celui-ci permet d'envoyer deux PWM de 3.3V d'amplitude et d'en contrôler les rapports cycliques. La fréquence de découpage a été revue à la hausse, passant de 1kHz dans nos premiers essais à 10kHz sur la version finale. Sur les deux voies, deux filtres passe-bas à 340Hz en cascade ont été ajouté dans le but de conserver essentiellement la valeur moyenne du signal (atténuation de la fréquence de découpage supérieure à 60dB). La résolution des PWM hardware étant meilleure que 10ns, on dispose donc d'une résolution de 13 bits de la pleine échelle, soit 0,03mm amplement suffisants pour notre application. Le schéma électrique final a été implémenté sur PCB (Cf. Figure 2.17 pour le schéma électrique).

Ce système permet donc d'envoyer à la table une séquence de tensions d'amplitudes

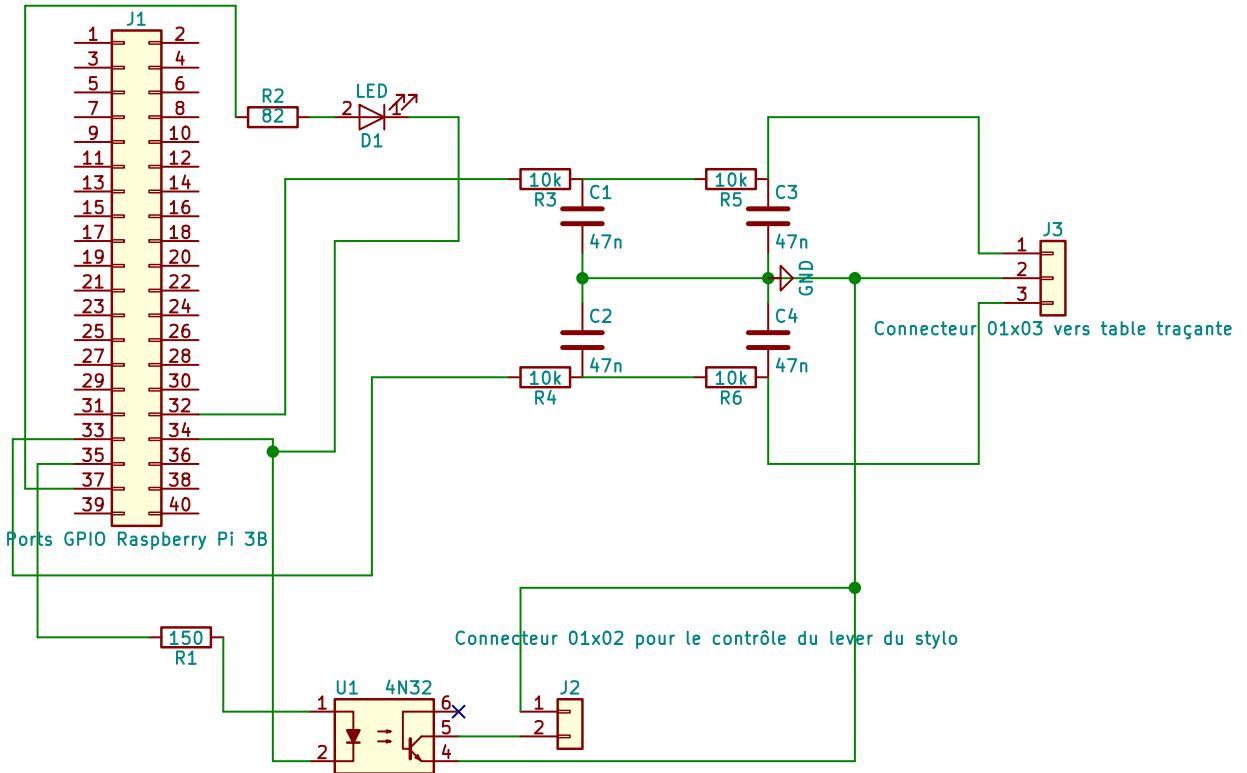


FIGURE 2.17 – Schéma électrique final

contrôlées entre 0 et 3.3V. En effet en contrôlant le rapport cyclique et sachant que la tension est fixée à 3.3V, les deux valeurs moyennes X et Y en sortie des filtres passe-bas sont contrôlées (Cf. Figure 2.18).

Le choix du calibre en V/cm s'est fait dans l'optique d'avoir l'accès à l'ensemble de la table traçante avec le stylo. C'est donc le calibre 0.1V/cm qui a donc été retenu sur les deux voies. Ainsi le stylo a accès à une plage de 33 cm sur x et sur y.

L'interrupteur contrôlant la montée et descente du stylo a été fait à l'aide d'un optocoupleur[4]. Un simple transistor MOS aurait été suffisant, mais l'optocoupleur a l'avantage de protéger la patte du Raspberry Pi. Dans le cas où le transistor fond, la patte se retrouverait directement à 15V, mais avec l'optocoupleur c'est une diode qui se trouve à cette patte et non directement le transistor et donc les 15V. En théorie il permet aussi de découpler la masse du Raspberry de la masse de la table, mais dans ce système il y aura équivalence entre les deux masses. Cependant cette protection sera toujours assurée.

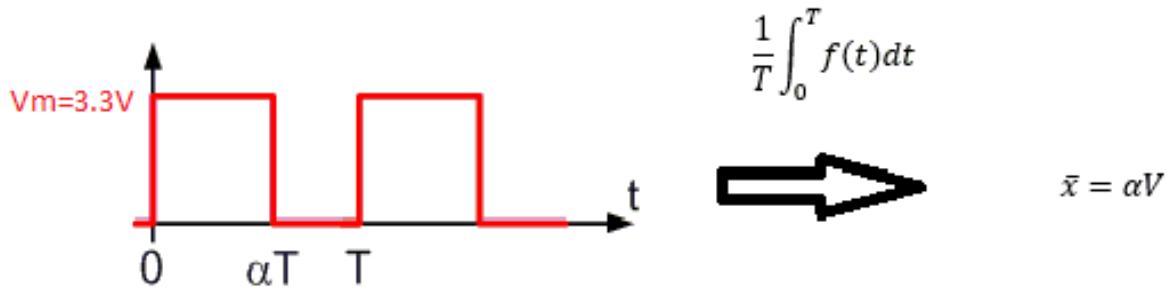


FIGURE 2.18 – Contrôle des tensions via le rapport cyclique

2.2.3 Algorithme d'écriture

Deux pattes du Raspberry produisent donc deux PWM dont le rapport cyclique est contrôlé à l'aide de Python qui contrôle également le niveau logique haut ou bas d'une troisième.

Les fonctions d'écriture basiques ont pour objectif de remplir une grille droite, de taille égale à 9 cm et dont l'origine est la même que celle de la table, c'est-à-dire la position du stylo quand il reçoit deux tensions nulles. La gestion de ces contraintes, permettant aux fonctions d'écriture de s'adapter dans d'autres situations envisageables, est détaillée dans la section suivante. En entrée, les fonctions d'écriture récupèrent la grille résolue numériquement par le reste de la chaîne du système sous forme d'un tableau.

Il faut deux types de fonctions : les fonctions d'écriture de chiffres (*un*, *deux*, ..., *neuf*) et une fonction qui positionne le stylo au centre de la case que l'on souhaite remplir. La fonction *position* prend en entrée l'indice de la ligne et de la colonne et retourne deux tensions X et Y images des distances *x* et *y* qui sont les coordonnées du centre de la case indiquée. Cela est possible du fait de la connaissance de la taille de la grille. A chaque écriture de chiffre, cette fonction ajoutera donc un offset sur les deux voies afin de gérer la position du stylo dans la grille.

Pour les fonctions d'écriture des chiffres, il convient de décomposer chacun de ces chiffres en barres droites. Chacun de ces traits pourra être tracé en envoyant un nouveau couple de tensions sur X et Y. Cependant si X et Y évoluent ensemble dans le cas d'un trait ni horizontal, ni vertical, il est essentiel qu'ils évoluent à la même vitesse. Or ce n'est pas directement le cas avec la table traçante. Par exemple si l'on part de (0,0) avec une consigne (0.5 ; 5) cm, le stylo va faire le déplacement sur *x* plus rapidement que sur *y* et ce ne sera pas une droite. Par conséquent il faut donc gérer plus finement les deux consignes. La solution est de ne pas envoyer directement la consigne finale mais de leur envoyer une série de consignes pour progresser simultanément jusqu'à la valeur finale. Pour cela créer une base de temps commune qui va de 0 à 1 en *n* points se révèle judicieux. En multipliant

nos deux consignes par ce vecteur, on obtient le comportement voulu. Un vecteur haut sera également créé sur la même base de temps pour signaler si le stylo doit être levé ou baissé. Le programme crée donc 3 vecteurs pour chaque trait : l'un contenant les positions successives sur x , l'autre sur y et le dernier binaire indiquant la levée ou baissée du stylo. Ensuite il suffit de concaténer ces trois vecteurs pour chaque trait du chiffre. La fonction d'écriture du chiffre renvoie donc deux séquences de tensions et une séquence binaire, les trois avec une base de temps commune (Cf. Figure 2.19)

Cette base de temps est liée au temps d'implémentation du rapport cyclique. Pour passer de la tension au rapport cyclique de la PWM, tous les termes en tension sont divisés par la tension V - ici 3,3V - : cela est juste un calcul numérique. Mais chaque implémentation d'un nouveau rapport cyclique prendra un temps dt . Sachant que chaque trait s'exprime par un vecteur de n valeurs et que l'on a plusieurs traits dans chaque chiffre, le temps de tracé d'un chiffre sera donc $T = Nb_{traits} \times n \times dt$. C'est donc la valeur de n qui détermine le temps d'écriture de la grille.

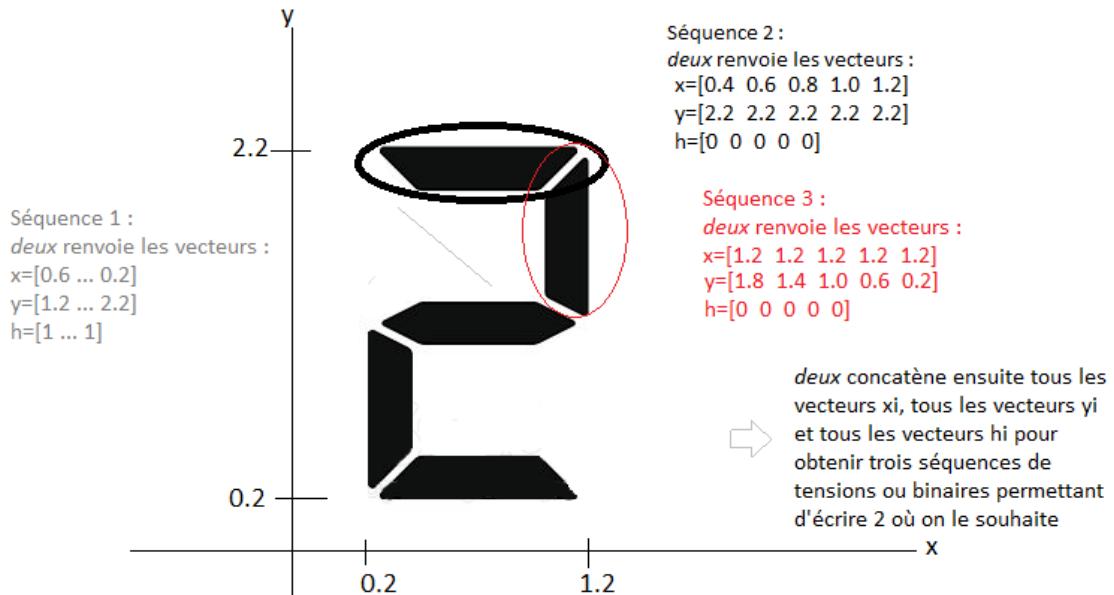
Le figure 2.19 résume ces étapes. Le raisonnement est en tension, car pour passer des tensions aux longueurs, il suffit de multiplier ou de diviser par le calibre.

À l'aide de deux boucle, l'une sur les lignes et l'autre sur les colonnes, le stylo parcourt donc de haut en bas et de gauche à droite l'ensemble de la grille. A chaque case le programme fait appel à la bonne fonction d'écriture parmi les neuf possibles à l'aide de la grille numérique résolue. Les cases initialement remplies sont signalées dans la grille numériques et le programme passe à la case suivante dans ce cas-là.

2.2.4 Gestion des contraintes

Dans cette étape, plusieurs contraintes doivent être prises en compte. Tout d'abord l'utilisateur doit pouvoir poser la grille à résoudre où il le souhaite sur la table traçante et avec l'orientation qu'il souhaite. De plus la taille de la grille peut varier d'une grille à l'autre. Le système conçu répond à toutes ces contraintes. Lors du traitement d'image, il récupère l'origine de la grille par rapport à la table traçante, l'angle d'orientation et sa taille afin de les fournir aux fonctions d'écriture.

Origine Les fonctions d'écriture retournent des séquences de tensions contrôlant la position et la levée du stylo. Ces tensions de positions sont calculées par rapport à la position initiale du stylo, c'est-à-dire l'origine de la table traçante. Récupérer l'origine de la grille permet d'en calculer les tensions images, il ne reste plus qu'à ajouter ces offsets aux séquences de tensions sur x et y pour que l'origine de la grille devienne la nouvelle position initiale du stylo.



* La séquence ne démarre pas en (0,0) mais en (0.6, 1.2) car ce sont les coordonnées renvoyées par la fonction position qui placent le stylo initialement au centre de la case. On a sommé ces coordonnées à tous les termes sur x et y.

FIGURE 2.19 – Étapes d'écriture

Orientation Les séquences de tensions que retournent les fonctions d'écritures permettent d'écrire les nombres droits dans un repère cartésien défini par les axes x et y de la table dont l'origine est désormais l'origine de la grille. Ce problème est surmontable en considérant le repère $(0, u_x, u_y)$ de la grille, qui possède la même origine mais est orienté avec l'angle de la grille par rapport au repère initial. Les équations de changement de bases permettent de ré-exprimer tous les points dans la base initiale. Les images en tensions de ces nouvelles valeurs tiennent compte de l'orientation de la grille. (Cf. 2.20)

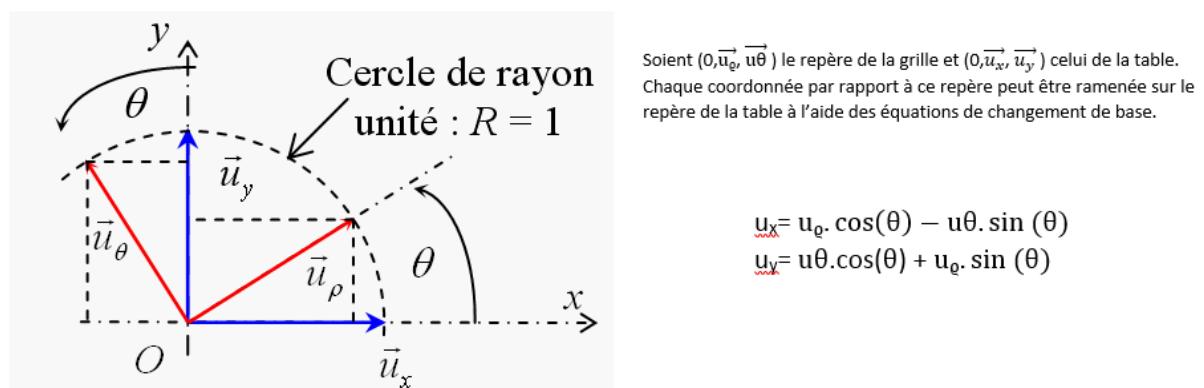


FIGURE 2.20 – Schéma du changement de base

Taille Les fonctions d'écritures sont conçues dans le but d'écrire une grille dont chaque case mesure 1cm de côté. Il s'agit de notre grille de référence qui est uniquement théorique. Elle simplifie l'écriture de ces fonctions car chaque raisonnement est fait sur des cases de 1cm. Le traitement d'image renvoie la taille de la grille. Les fonctions utilisent la formule $agrandissement = taille/9cm$ pour avoir le rapport entre la taille de l'image réelle et la taille de l'image théorique de référence. Sachant que les tensions sont proportionnelles aux déplacements, les multiplier par ce facteur d'agrandissement sans modifier l'origine permet de les adapter à la taille réelle.

Calibrage Pour déterminer la taille, l'angle et l'origine il faut préalablement avoir fait un calibrage. Pour cela le programme trace le plus grand rectangle possible et en prend une capture. En traitant cette image, le système récupère l'origine du cadre et comme la taille du plus grand rectangle est une donnée connue (27.5, 18.3 cm), il peut faire une correspondance pixel \leftrightarrow cm à partir de laquelle on détermine toutes les informations voulues.

2.2.5 Remplissage du Sudoku

La valeur de n règle la vitesse, le compromis entre vitesse et lisibilité est pour $n = 400$. Les figures 2.21 et 2.23 montrent une grille résolue et complétée par le système.

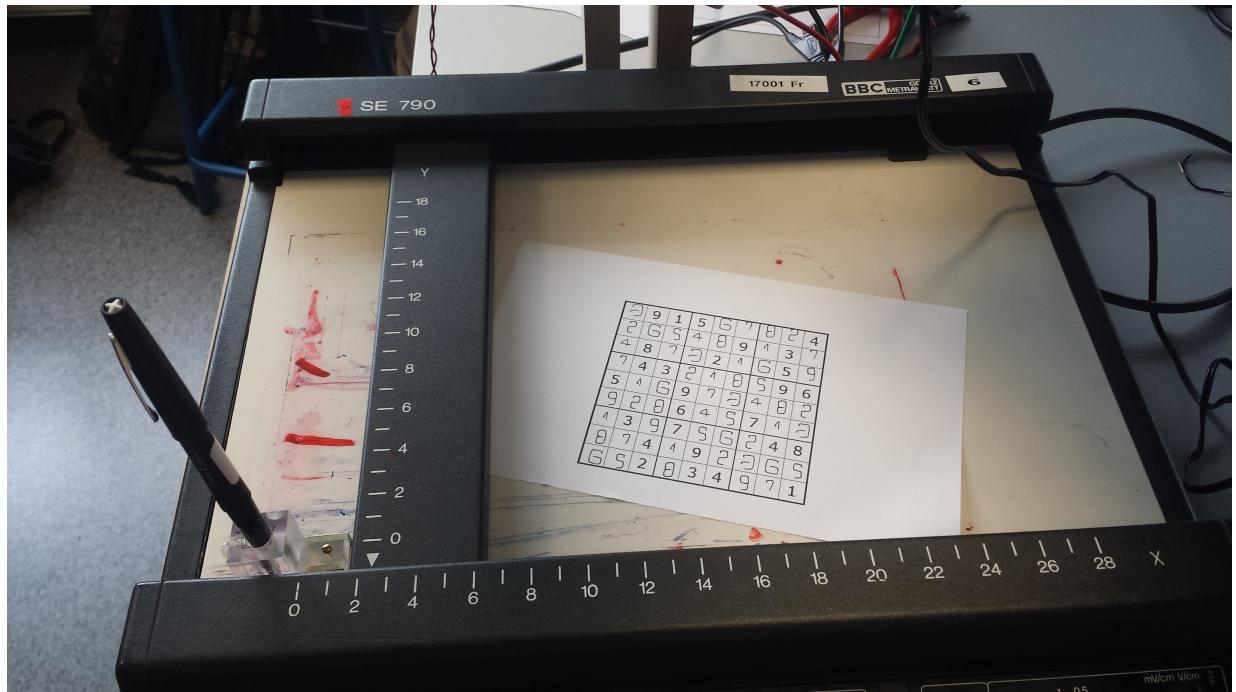


FIGURE 2.21 – Grille Résolue par le système

On remarque que les contraintes énumérées plus tôt, le placement, la taille et l'ori-

tation quelconques, ont toutes été prises en compte.

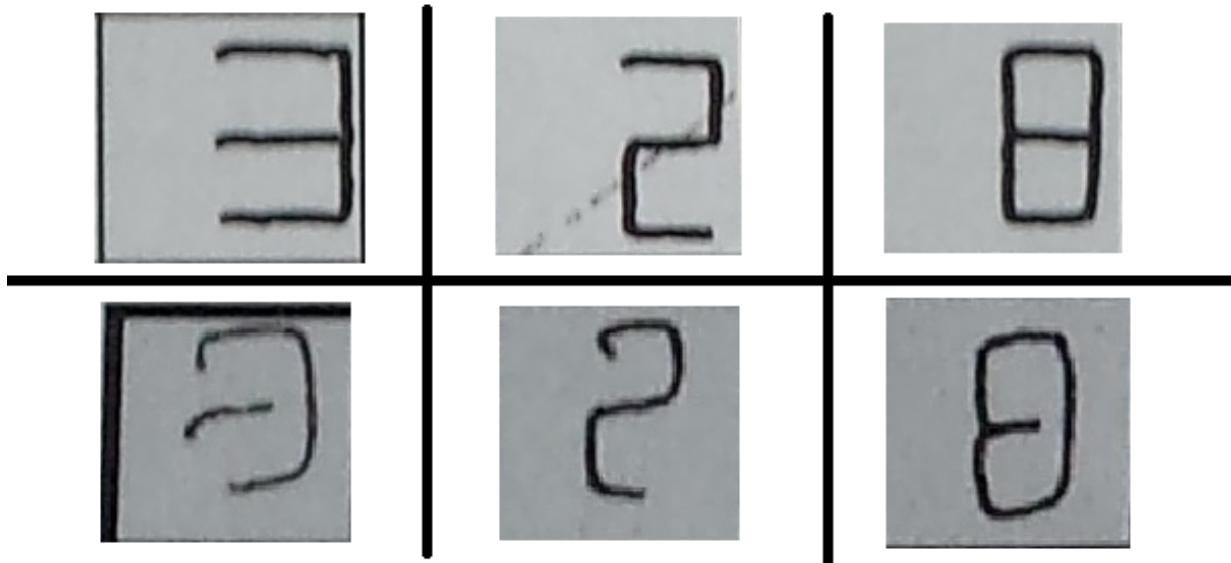


FIGURE 2.22 – Chiffres écrits pour $n=4000$ (haut) et $n=400$ (bas)

On peut voir sur la figure 2.22 l'influence du choix de la valeur de n . Les chiffres du bas ($n = 400$) ont été tracés beaucoup plus rapidement et ils sont beaucoup moins nets que ceux du haut pour lesquels $n = 4000$. On peut également voir qu'il serait intéressant de modifier le 3 et le 8 qui ne sont pas très esthétiques. Cette modification ne nécessite cependant pas de changer la valeur de n , il est possible de se contenter de modifier les fonctions d'écriture des chiffres concernés. Au delà de l'esthétisme, on ne peut pas aller en dessous de $n = 300$ car si l'on essaie de faire passer le rapport cyclique de 0 à 1 avec un vecteur de moins de 300 points, on observe à l'oscilloscope qu'en réalité il n'atteint jamais 1. Pour $n = 400$ le système remplit la grille en approximativement 3 minutes.

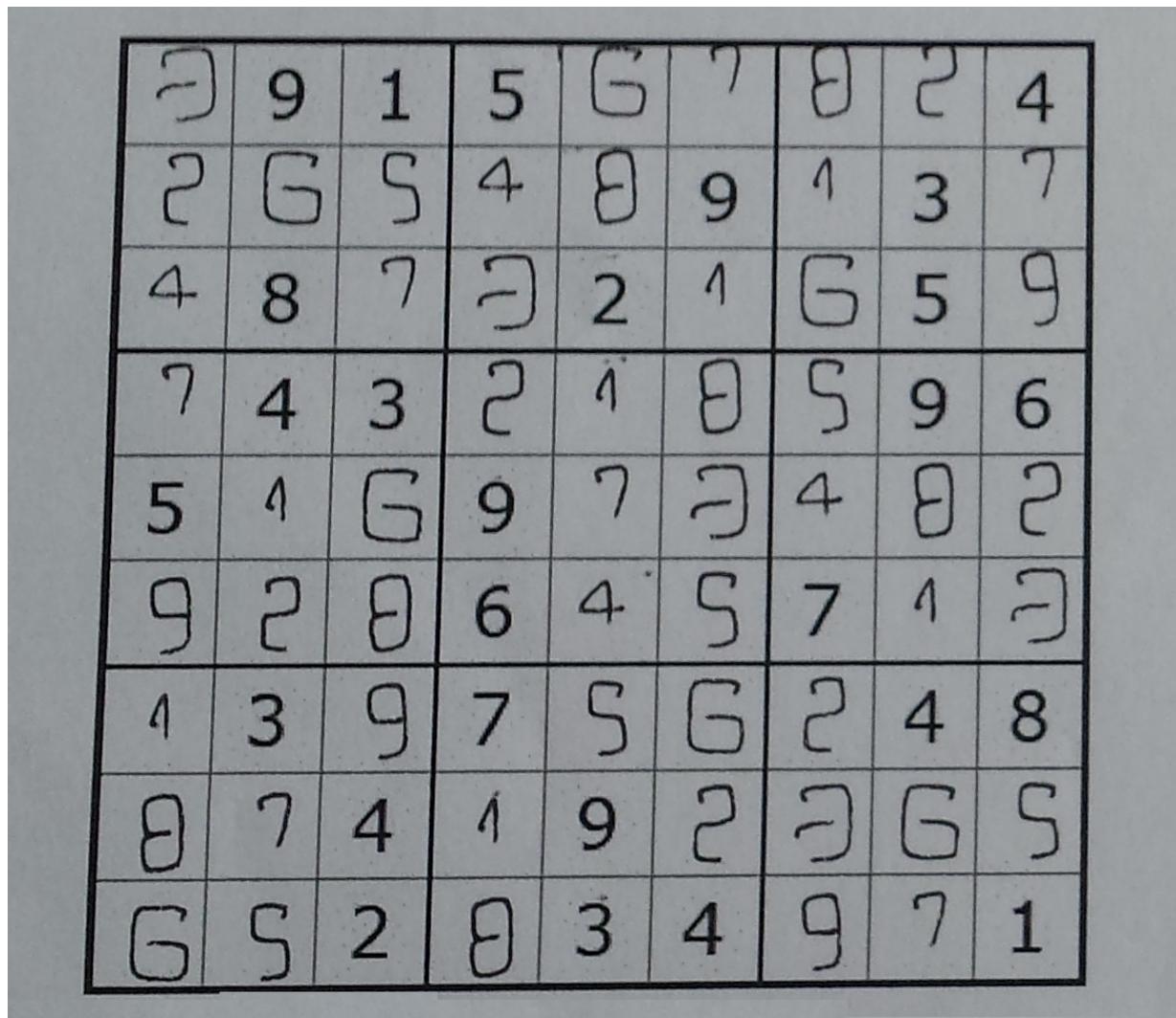


FIGURE 2.23 – Grille Résolue par le système pour n=400

Chapitre 3

Bilan du travail

3.1 Réalisation finale

La réalisation finale de notre projet respecte le critère d'autonomie de notre cahier des charges. La seule action nécessaire à l'exécution du dispositif est l'exécution d'une ligne de commande en mode super utilisateur (contrôle des ports GPIO) : `sudo python3 main.py`

La grille de sudoku peut avoir une position quelconque sur la surface de la table traçante. La taille du sudoku s'adapte aussi automatiquement.

Notre réalisation finale comporte toutefois certains points d'amélioration. Un éclairage ¹ de la table relativement uniforme donne des résultats de reconnaissance optimaux.

Notre algorithme gère actuellement les sudokus 9x9 mais pourrait être facilement modifié pour des tailles de grille différentes. Il permet la détection et le traitement des grilles multiples sur une même image.

La détection des grilles devient toutefois moins efficace lorsque les lignes intérieures deviennent très fines. En effet, les contours alors détectés peuvent être non fermés, rendant l'algorithme de détection de grille inopérant. Nous rencontrons aussi des problèmes au niveau de la détection de la grille pour les orientations trop proches de 45° par rapport à l'horizontale.

En résumé, malgré quelques difficultés aujourd'hui levées pour dompter la table traçante, la chaîne complète est, en dehors des derniers points soulevés, fonctionnelle. Nous sommes ainsi relativement satisfaits du résultat, d'autant plus qu'il résulte d'un travail relativement homogène de chaque membre du groupe.

1. Avec les pré-réglages de la caméra, il apparaît une légère surexposition sur l'image capturée par rapport à notre perception de la scène. Cela permet de travailler dans un environnement un peu sombre tel qu'une pièce ; nous laisserons donc ce réglage en l'état.

Table traçante Servogor 790

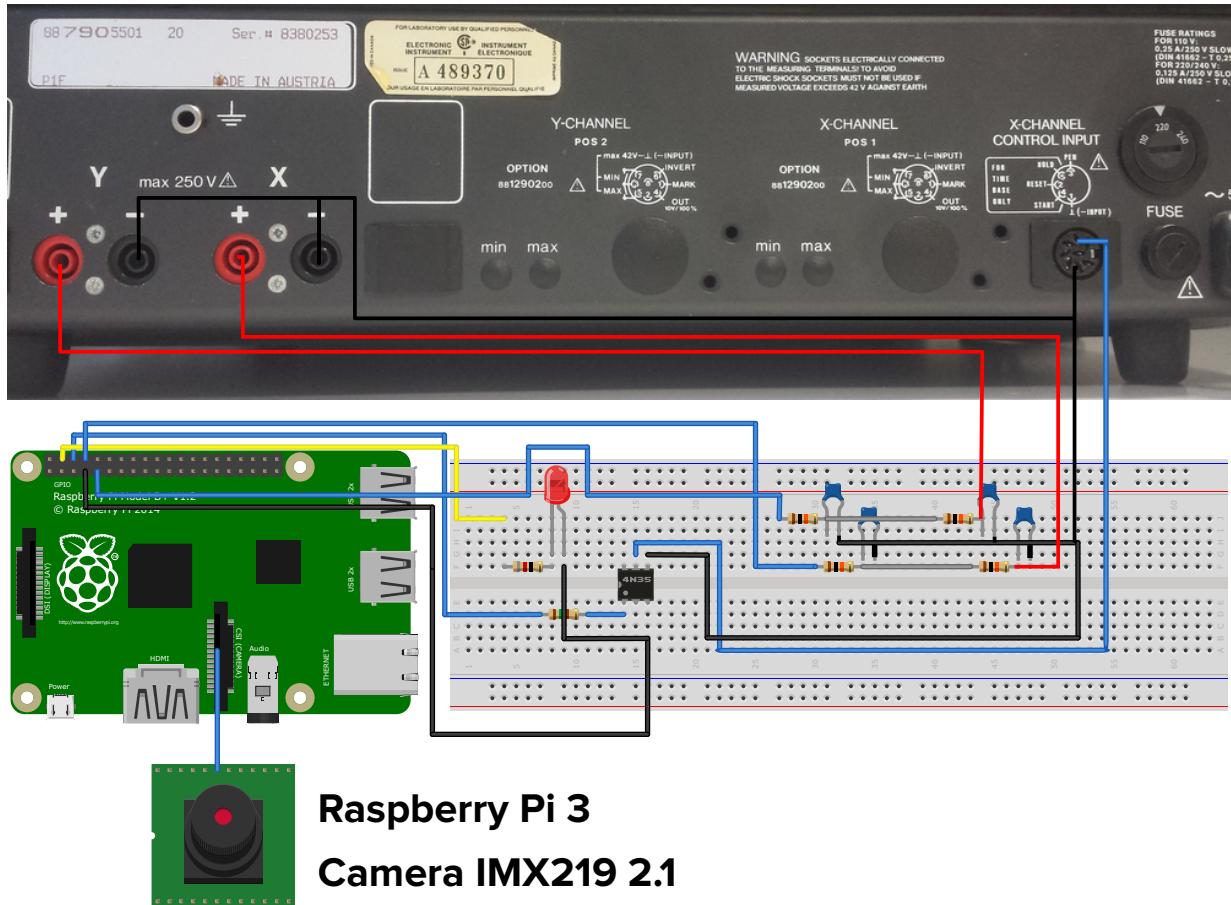


FIGURE 3.1 – Schéma électrique global. Réalisé via le logiciel *Fritzing*.

La connecteurs banane situés sur la gauche de la table traçante permettent de contrôler le déplacement du stylo, tandis que le connecteur MIDI à droite contrôle la levée du stylo.

3.2 Évaluation du coût de fabrication

Matériel	Quantité	Coût total
Raspberry Pi 3B	1	40 €
Camera Pi 2.1	1	32 €
Adaptateur HDMI mâle / VGA femelle	1	≈ 6 €
Table traçante Servogor 790	1	Non Communiqué
Bois (hors découpe)	3251 cm ²	Non Communiqué
PCB et composants	-	≈ 10 - 15 €

TABLE 3.1 – Coût du matériel utilisé

Nous estimons le coût du projet à une centaine d'euros.

Bibliographie

- [1] T. E. de CAMPOS, B. R. BABU et M. VARMA. *Character recognition in natural images*. Fév. 2009. URL : <http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/>. (consulté le 26.02.2018).
- [2] John CANNY. “A Computational Approach to Edge Detection”. In : *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986), p. 679–698.
- [3] Scott ELLIS. *pwm.py*. 2017. URL : <https://github.com/scottellis/pwmpy/blob/master/pwm.py>.
- [4] *Optocoupler, Photodarlington Output, High Gain, With Base Connection*. Optocoupleur 4N32. Vishay Semiconductors. URL : <https://compelectronic.fr/pdf/4n32.pdf>.
- [5] OpenCV ORG. *OpenCV 3.4.1 Modules*. URL : <https://docs.opencv.org/3.4.1/>.
- [6] OpenCV ORG. *Understanding k-Nearest Neighbour*. Nov. 2014. URL : https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_ml/py_knn/py_knn_understanding/py_knn_understanding.html. (consulté le 13.03.2018).
- [7] Adrian ROSEBROCK. *Raspbian Stretch : Install OpenCV 3 + Python on your Raspberry Pi*. Sept. 2017. URL : <https://www.pyimagesearch.com/2017/09/04/raspbian-stretch-install-opencv-3-python-on-your-raspberry-pi/>.
- [8] Uwe SCHÖNING. *Algorithmics in Exponential Time*. 2008. URL : <http://faculty.cs.tamu.edu/chen/courses/637/2008/pres/fan.pdf>. (consulté le 10.04.2018).
- [9] Jules SVARTZ. *TP : Algorithmes de Backtracking et résolution de Sudoku*. 2015. URL : http://perso.numericable.fr/jules.svartz/prepa/IPT_spe/TP_sudoku.pdf. (consulté le 21.03.2018).
- [10] Jumpnow TECHNOLOGIES. *Using the Raspberry Pi hardware PWM timers*. 2017. URL : <http://www.jumpnowtek.com/rpi/Using-the-Raspberry-Pi-Hardware-PWM-timers.html>.
- [11] XY-Yt Analog Compact Recorder. Servogor 790. Kipp & Zonen. URL : https://www.meter.hu/adatlap/regisztralo/pdf/kipp_brochure_se790_1024.pdf.