FALCONNIER Pierre

PAYSAN Jérémie

GIRARDEY Martin

LIORET Nathan

Rapport du Projet Collectif SICOM 2A

Analyse, résolution et remplissage automatique d'une grille de Sudoku

Grenoble INP - Phelma

Grenoble INP - ENSE^3

Université Grenoble Alpes

Mai 2022

Sommaire

1 Introduction

1.1 Contexte du projet

Dans le cadre de nos études en deuxième année d'école d'ingénieur à Grenoble INP - Phelma en filière SICOM, nous avons l'opportunité de réaliser un projet dit "collectif" sur un sujet de notre choix parmi ceux proposés.

Celui que nous avons choisi consiste en l'élaboration d'un système permettant la résolution automatique de grilles de Sudoku. L'utilisateur n'a qu'à poser une grille non complétée sur la table traçante et celle-ci va être complétée automatiquement.

Ce sujet n'est pas nouveau puisque une équipe d'étudiants s'était déjà penchée dessus l'année dernière, également dans le cadre du projet collectif. Nous sommes donc partis de leur travail. Mais, ne comprenant pas toujours très bien ce qu'ils avaient fait et souhaitant parfois faire les choses différemment, nous avons refait une grande partie de leur travail, qui n'en reste pas moins très conséquent.

1.2 Cahier des charges

Nous définissons ici le cahier des charges du système. Il avait déjà été réalisé très précisément l'année dernière et seuls quelques détails changent. Nous nous sommes appuyés dessus tout au long du projet.

L'objectif est d'obtenir la grille de sudoku résolue sans erreur en un temps assez court ($<$ 2 minutes). A priori, la majeure partie du temps de résolution de la grille sera dédiée à l'écriture des chiffres. Il faudra tout de même s'assurer de la lisibilité des chiffres écrits par la machine, en étant le plus précis possible. La résolution de la grille devra être effective quelque soit la dimension de la grille de Sudoku, dans les limites des dimensions de la table traçante (jusqu'à 20 cm de côté au maximum) et quelque soit l'orientation de la grille posée sur la table traçante. La dimension de la grille doit être suffisamment grande pour assurer la lisibilité des chiffres. Les programmes de traitement d'images et de résolution devront être suffisamment généraux de manière à pouvoir les tester sur des cas simples sans avoir ensuite à les réécrire entièrement pour les cas plus complexes.

1.3 Matériel et choix de conception

Pour mener à bien ce projet, nous avons utilisé différents composants. Une table traçante nous a été fournie ainsi qu'une structure en bois. La table est pilotée suivant deux axes (X et Y) au travers d'une tension que l'on peut calibrer (intervalle de tension, offset).

Nous utilisons une Raspberry Pi 4 branchée à un SSD pour exécuter l'ensemble des programmes nécessaires, ainsi qu'une Arduino Mega 2560 pour commander la table traçante.

Nous disposons aussi d'une caméra (v2) pour prendre en photo la grille, et de deux convertisseurs numérique-analogiques (CNA) pour relier l'Arduino à la table. Et enfin un bouton poussoir pour démarrer le système.

Au cours du projet, nous avons aussi fabriqué une petite plaque électronique que l'on branche à l'Arduino et qui permet, entre autres, de piloter la levée du stylo et de partager des informations au travers d'une LED (voir partie 2.3.4).

Nous avons fait le choix du langage Python pour l'élaboration de nos codes du fait de nos affinités pour ce langage que nous utilisons au quotidien, notamment pour le traitement d'image. Nous sommes toutefois conscients de l'impact de ce choix sur les performances du système.

1.4 Organisation et répartition du travail

Le projet peut se diviser en plusieurs parties distinctes, permettant une répartition efficace du travail dès le début : Jérémie Paysan et Pierre Falconnier se sont penchés sur toute la partie Traitement d'Image (acquisition de l'image, extraction des cases, orientation, reconnaissance des chiffres). Nathan Lioret s'est occupé de la résolution du Sudoku, de l'allure des chiffres et de la communication Raspberry-Arduino. Enfin, Martin Girardey a travaillé sur toute la partie électronique/mécanique (pilotage de la table traçante par l'Arduino, installation des bibliothèques sur la Raspberry, fabrication de la plaque électronique et montage).

C'est du moins la répartition initiale du travail, puisque la partie Traitement d'Image était terminée bien avant les parties électronique/mécanique et communication Raspberry-Arduino. En effet, nous avons "perdu" près de 3 séances à cause de grandes difficultés à installer les bibliothèques Python nécessaires sur la Raspberry. Ainsi, durant les dernières séances, nous nous sommes tous concentrés sur la partie Électronique/Mécanique et Communication Raspberry-Arduino, sauf Jérémie qui a commencé la réalisation du circuit électronique sur KiCad.

2 Réalisation et déroulé du projet

2.1 Traitement d'image

2.1.1 Acquisition de l'image

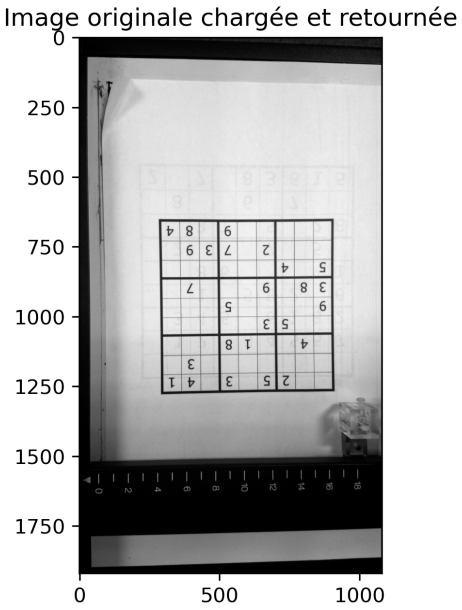
L'appuie sur le bouton poussoir relié au port 10 (GPIO) de la carte Raspberry Pi permet de lancer le programme python de résolution de la grille. Après appui sur le bouton, une photo de la grille est prise, la caméra étant fixée au deçu de la table traçante.

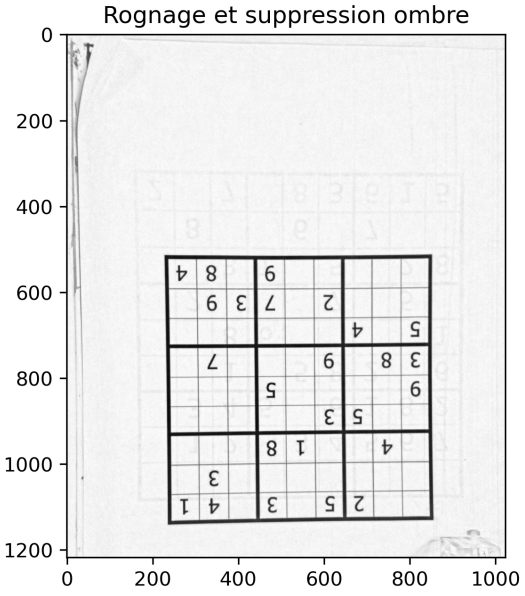
2.1.2 Extraction des cases de la grille de sudoku

A partir de l'image acquise, l'objectif est d'extraire la matrice des chiffres du sudoku pour la résoudre, l'angle d'inclinaison de la grille et la position de ses coins pour tracer au bon endroit et dans le bon sens les chiffres de la solution sur le papier. Les images sont manipulées via les bibliothèques Open-CV et Numpy.

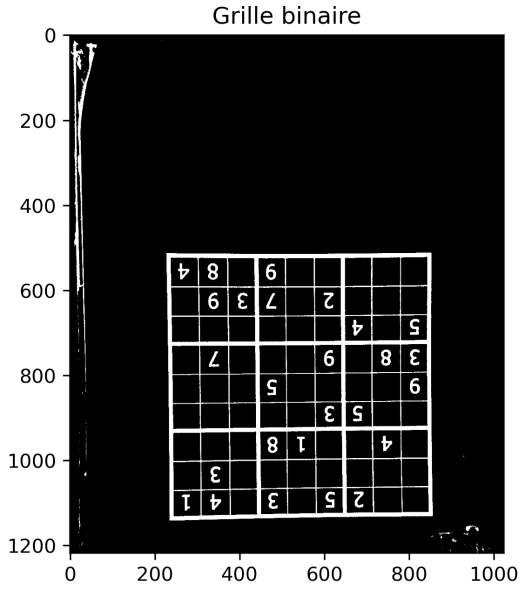
L'image sauvegardée est chargée en niveaux de gris (codés sur 8 bits) et tournée dans le bon sens (orientation de référence que l'on considère arbitrairement comme à l'endroit). Elle est rognée pour ne conserver que la zone d'intérêt où se situe la grille.

On applique un premier traitement pour augmenter le contraste et effacer de potentielles zones d'ombres : une dilatation suivie d'un filtre médian pour récupérer le fond, que l'on soustrait à l'image originale. On a alors le texte dont les niveaux de gris sont proches de 255. On inverse le résultat et on le normalise. Dans l'exemple choisi, la grille de sudoku a volontairement été placée à l'envers.

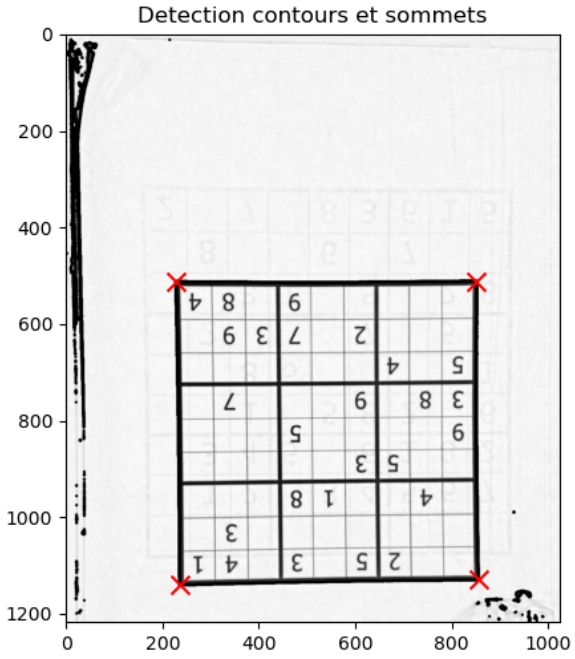


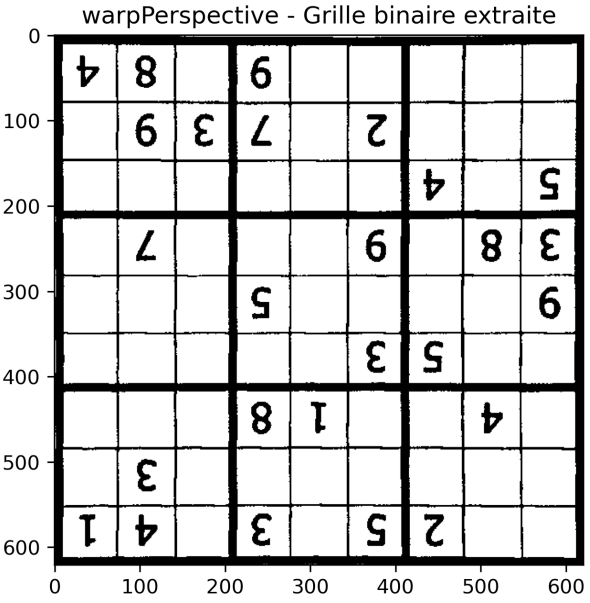


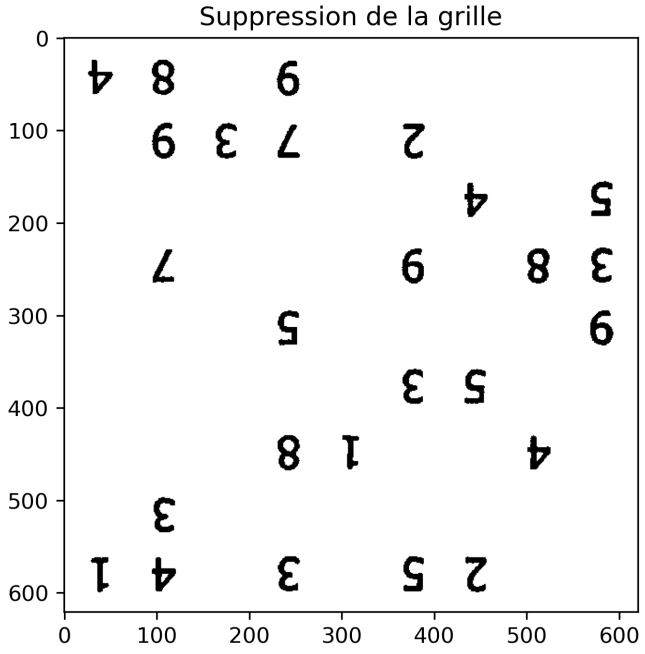
Pour faciliter la suite du traitement, l'image est binarisée. Le choix du seuillage est délicat car il impacte toute la chaîne de traitement en aval. Un seuillage automatique par la méthode d'Otsu donne des résultats satisfaisants mais variables selon les grilles traitées. Pour la détection de contours qui suit, il faut utiliser l'inverse de l'image binaire : les pixels dont le niveau de gris est inférieur au seuil sont placés au niveau 255, les autres au niveau 0.



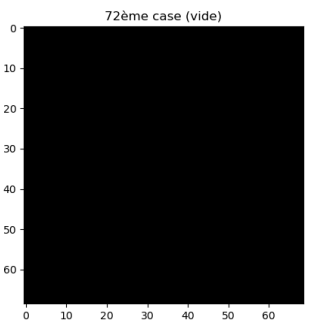
Le contour externe de la grille de sudoku étant carré, il est définie par la position de ses sommets. La fonction findContours d'open-cv permet l'acquisition de tous les contours externes des objets de l'image. On trouve celui de la grille comme étant le plus grand contours définie par quatre sommets. La grille est maintenant localisée sur l'image : on connaît ses sommets et sa taille. Ces données sont utilisées en entrée de la fonction getPerspectiveTransform [1] qui calcul la matrice de transformation nécessaire à la fonction warpPerspective qui permet de générer une image de la grille extraite. La position des sommets permet aussi de calculer l'angle entre le segment supérieur de la grille est l'horizontale de l'orientation de référence.







Il faut à présent extraire le contenu des cases. Les lignes noires de la grille sont effacées de proche en proche (par "inondation") en partant des bords. L'image est divisée en 81 images de même tailles, qui correspondent aux cases de la grille. Les cases vides sont identifiées par le proportion de pixels blancs



2.1.3 Orientation de la grille

La grille de sudoku étant placée sur la table traçante dans une orientation quelconque, l'image de la grille extraite n'est à l'endroit qu'à un multiple de 90° près. Il est nécessaire de trouver la bonne orientation pour connaître l'angle total relatif à l'orientation de référence mais aussi pour procéder à la reconnaissance des chiffres.

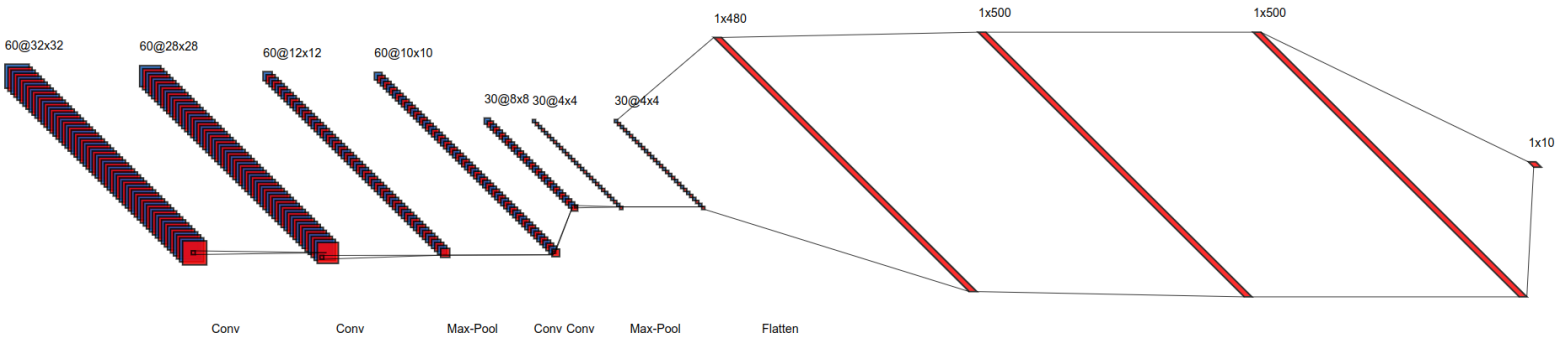
En partant de l'hypothèse (vérifiée) que les chiffres imprimés ont généralement leur hauteur supérieure à leur largeur, il est possible d'écarter les deux orientations pour lesquelles les chiffres seraient à l'horizontale. C'est le rôle de la fonction orientation qui, pour chaque case de la grille contenant un chiffre, va incrémenter ou décrémenter un compteur selon que la dimension du chiffre suivant l'axe x est plus grande ou plus petite que la dimension suivant l'axe y de l'image. En fonction du signe du compteur, on en déduit si l'orientation de la grille est selon l'horizontale ou la verticale. Effectuer le test sur tous les chiffres de la grille permet de limiter les erreurs.

Les cases sont à présent dans le bon sens à 180° près. Pour déterminer si une rotation est nécessaire, les cases identifiées comme non-vides sont données en entrée à un réseau de neurones

2.1.4 Réseaux de neurones : reconnaissance de l'orientation et des chiffres

<u>Choix du modèle</u>

Pour la reconnaissance des chiffres, notre choix de modèle de prédiction s'est naturellement porté vers le réseau de neurone convolutif LeNet [2], proposé par Yann Lecun dans le cadre de la reconnaissance de chiffres manuscrits. Par soucis de simplicité, nous avons aussi utilisé une réseau LeNet pour la reconnaissance de l'orientation des chiffres. Nous avons implémenté le modèle en reprenant un code publié sur github.com [3] utilisant la bibliothèque python Keras (API qui s'appuie sur TensorFlow).



Description de l'architecture

Le réseau est dit convolutif. En effet, ce qui fait l'efficacité de cette architecture pour le traitement d'image sont les couches de filtres convolutifs en amont du réseau de neurone. Ils permettent de réduire le nombre d'entrées du réseau en sélectionnant l'information essentielle.

L'image est filtrée par 60 filtres convolutifs à deux reprises, puis on effectue une opération de max-pooling qui consiste à conserver uniquement le pixel de plus haute intensité dans un certain voisinage. On concentre l'information (à ce stade les "images" sont de dimension 10x10). A nouveau, elles passent par deux couches convolutives (de 30 filtres cette fois) puis une max-pool. Les 30 images de dimensions 4x4 sont alors étirées en un vecteur de longueur 480 placé en entrée d'un réseau de neurones à deux couches cachées de taille 500. Le réseau donne alors en sortie un vecteur de taille 10 qui permet de classifier l'image en entrée.

La fonction d'activation choisie est la fonction ReLu [4], sauf entre la dernière couche cachée et la couche de sortie où l'on utilise la fonction Softmax [5]. La fonction Softmax renvoie un vecteur de 10 réels compris entre 0 et 1. La classe du chiffre de l'image d'entrée correspond alors à la position dans le vecteur de la valeur la plus proche de 1.

Pour l'entraînement, la fonction de coût sera l'entropie croisée, minimisée via l'algorithme d'optimisation Adam.

Données d'entraînement

Les chiffres de grilles de sudoku étant imprimés (et non pas manuscrits) nous avons utilisé un jeu de données d'entraînement adapté [6]. Bien que 0 ne soit pas présent sur une grille de sudoku, il a été plus simple dans la conception de notre code de le conserver. Le jeu de données compte 1 016 images pour chacun des chiffres (10 160 images au total), écrits dans des polices variées. Les images sont de résolution 128x128 et codées sur 3 canaux de 8 bits.

Pour le modèle de reconnaissance d'orientation, on ajoute aux données les images tournées de 180° et on leur associe un label correspondant. On considère que 6, 9, 8 et 0 avec le label "à l'endroit" car le 6 et 9 sont identiques à une rotation de 180°, et le 0 et 8 sont invariants par rotation de 180°. Ces chiffres ne permettent donc pas de savoir si une grille de sudoku est à l'endroit ou non.

Prétraitements des images

Une résolution d'images de 128x128 est trop importante. Afin de réduire le temps d'entraînement et la rapidité du réseau, les images sont redimensionnées en 32x32. Elles sont ensuite converties en niveau de gris (codées sur 8 bis). Puis on procède à une égalisation d'histogramme avant de diviser la valeurs des pixels par 255 : les niveaux de gris sont donc représentés par des nombres flottants compris entre 0 et 1.

Entraînements et scores

On charge des données d'entraînement : on concatène les images dans une matrice et on crée un vecteur avec les labels associées. On entraîne le modèle sur 80% des données choisies aléatoirement (avec une répartition uniformes des classes). Les 20% restant permettront d'évaluer le modèle sur des données nouvelles. On augmente artificiellement le nombre de données à disposition en créant de nouvelle images à partir de celles que l'on possède via différentes transformations (légères) : rotations, translation, zoom.

On calcule le pourcentage de prédictions correctes des modèles sur les données de test.
Reconnaissance de chiffres : 94,69 %
Reconnaissance de l'orientation des chiffres : 98,89 %

On considère donc les modèles comme performants.

2.1.5 Retour au Sudoku

Il suffit maintenant de parcourir les images des cases du sudoku et d'appliquer la prédiction de l'orientation des chiffres. Mais avant la prédiction, les images doivent être prétraitées pour être les plus similaires aux images sur lesquelles le réseau s'est entraîné.

Sur les données d'entraînement, on calcul en moyenne qu'un chiffre occupe 75% de la hauteur de l'image et 50% de sa largeur. Pour chaque cases du sudoku contenant un chiffre, on isole ce chiffre et on calcul ses dimensions. On crée alors une nouvelle image où l'on place ce chiffre en choisissant les dimensions de l'image de sorte que les proportion 75% et 50% soient respectées. On centre alors le chiffre dans l'image en calculant son centre de gravité et en le transmettant au centre de l'image. Enfin, on redimensionne l'image en taille 32x32 pour pouvoir être donnée en entrée du réseau de neurones convolutif.

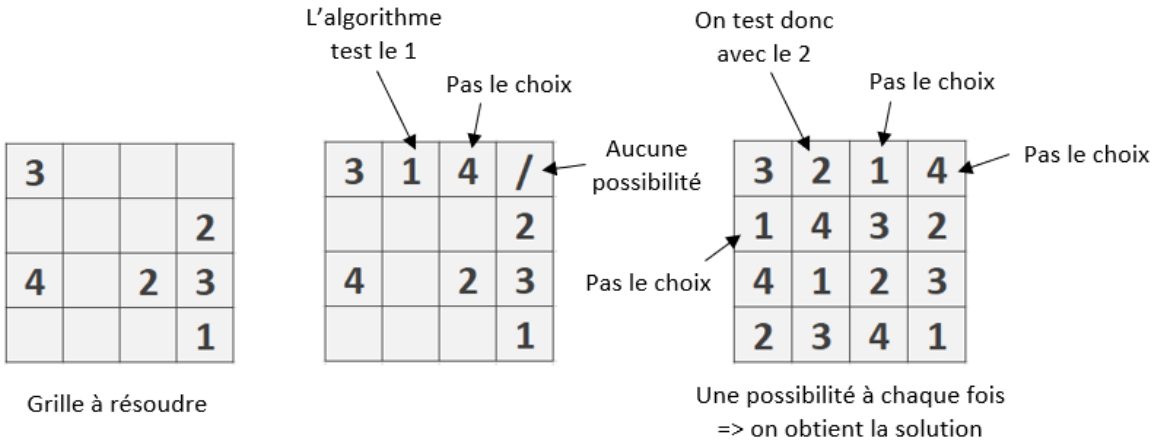
On applique le modèle au cases. Dans le cas où 2 chiffres ou plus sont prédit comme étant à l'envers, on tourne la grille de 180°. On prend en compte cette rotation dans le calcul de l'angle total relatif à l'orientation de référence. Les images des cases sont à présent à l'endroit, on peut appliquer le modèle de reconnaissances de chiffres sur les cases après les avoir prétraitées comme précédemment. On obtient un matrice de taille 9x9, contenant les chiffres prédit par le modèle à leur position correspondante (pour les cases vides, on prend le chiffre 0).

On a donc obtenue la matrice de la grille de sudoku, la position sur l'image de la grille ainsi que son angle total d'inclinaison.

2.2 Algorithme de résolution du Sudoku

Une étape clé dans la conception de notre système est la résolution du sudoku une fois que l'on a lu la grille et avant d'écrire la solution.

Pour cela, différentes méthodes existent (backtracking, stochastique…). Nous utiliserons la méthode classique de backtracking, la plus simple à mettre en place mais qui reste tout de même très efficace. Cette méthode intuitive consiste à tester toutes les possibilités, en revenant en arrière si l'on constate que l'on n'est plus sur la bonne voie (si une case ne dispose plus d'aucune possibilité) [8]. Si la grille est résoluble alors on s'arrête à la première solution trouvée (qui est la seule si la grille est bien faite). On implémente cette méthode au moyen d'une fonction récursive dans laquelle on recalcule à chaque fois les chiffres possibles pour les cases. La figure suivante illustre, sur une grille 4x4, quelques itérations avec la méthode du backtracking :



Nos prédécesseurs sur ce projet avaient déjà utilisé cette méthode du backtracking pour la résolution mais comme nous ne comprenions pas bien leur code et que nous souhaitions l'optimiser, nous l'avons refait entièrement. Nous avons aussi essayé de mettre en place la méthode stochastique [8]. Mais cela n'a pas marché et nous n'avons pas voulu insister car notre nouvel algorithme de résolution par backtracking est déjà presque 10 fois plus rapide que le leur (sur 1000 tests) et nécessite un temps de résolution de l'ordre du dixième de seconde, ce qui est largement négligeable devant les autres temps mis en jeu dans le système (traitement d'image et tracé surtout).

Cette méthode, comme elle teste toutes les solutions, peut de fait être assez lente si aucune solution n'existe. C'est typiquement ce qui arrive lorsque les chiffres ont mal été lus lors du traitement d'image. Pour pallier ça, il pourrait être intéressant de regarder, avant de tenter la résolution, si la grille "à l'air" est correcte (ie si aucun doublon n'est déjà visible sur les lignes, les colonnes et dans les blocs).

Il est aussi intéressant de noter que pour pouvoir effectuer nos tests, nous avons également mis en place la génération de grilles de Sudoku. La fonction ainsi créée utilise la résolution par backtracking, un peu modifiée toutefois, afin de vérifier l'unicité de la solution.

2.3 Ecriture de la solution

On veut écrire sur la grille papier la solution trouvée précédemment au moyen d'une table traçante. Nous l'avons vu, cette table est pilotée au travers de deux tensions (X et Y) et d'un interrupteur Z qui traduit la levée du stylo. Tout l'enjeu est de déterminer quelles tensions envoyer pour tracer les chiffres dans les cases comme on le veux.

2.3.1 Correspondance entre l'image issue de la caméra et les tensions envoyées à la table traçante

Le traitement d'image étant effectué sur une image, les coordonnées qu'il fournit sont exprimées en pixels et l'origine n'est pas la même que celle de la table traçante.

On a une simple relation linéaire entre la position en pixels et la position que l'on envoie à la carte Arduino, qui sera traduite à son tour en tension. Pour déterminer les coefficients de cette relation, en X et en Y, nous avons fait différentes mesures : la longueur de l'axe sur l'image (1531 en X et 923 en Y, en pixels) et l'intervalle de tension nécessaire pour parcourir entièrement ce même axe (3300 en X et 2330 en Y) nous permettent de déterminer le rapport d'échelle. Et la position de l'origine de la table sur l'image nous donne l'offset (50 en X et -45 en Y, en pixels).

Au final, si (x_{image}, y_{image}) sont les coordonnées d'un point dans le référentiel de l'image, alors (x_{table}, y_{table}) les coordonnées de ce même point dans le référentiel de la table, suive la loi suivante :

$$(x_{table}, y_{table}) = (\, 3300 \times (x_{image} + 50)/1531 \,,\, 2330 \times (y_{image} - 45)/923 \,)$$

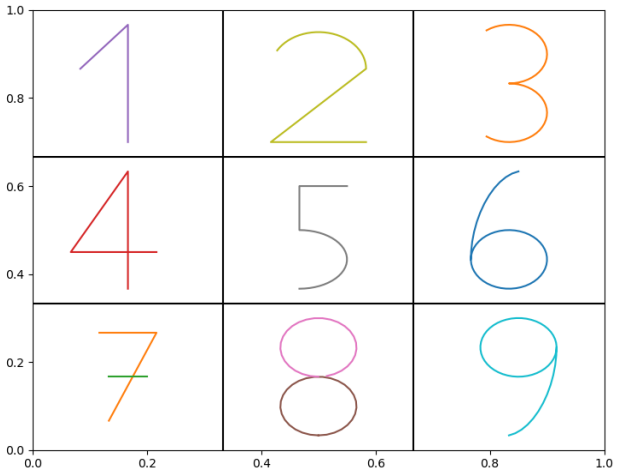
On considère que la taille d'une case sur l'image est la distance entre deux sommets (consécutifs) divisée par 9. On obtient la correspondance en tension en appliquant les mêmes coefficients multiplicateurs que précédemment. Un problème auquel nous faisons face et qui se voit lors de l'écriture, est "l'agrandissement des cases" dû aux bords larges sur certaines grilles, car nous regardons les coins externes alors que nous devrions regarder les coins internes.

On ajuste ensuite "à la main" ces coefficients pour assurer la meilleure précision d'écriture possible (on vérifie que les chiffres soient bien écrits au centre des cases et ne qu'ils ne dépassent pas sur les cases adjacentes). Le défaut de cette approche est que la relation devient fausse si les positions relatives de la table traçante et de la caméra changent.

2.3.2 Calcul des tensions associés aux chiffres

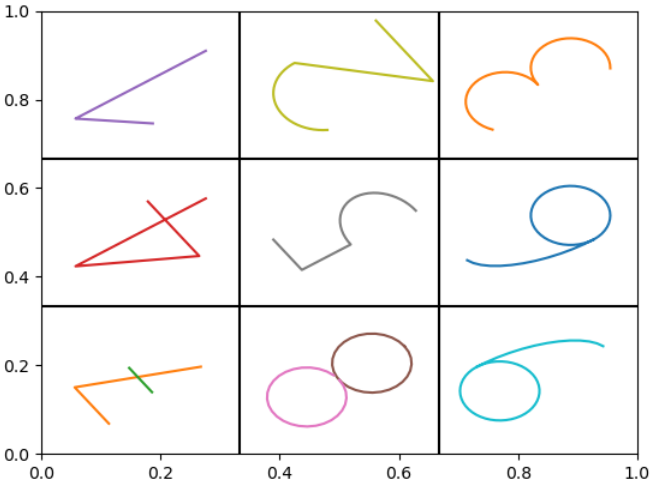
Le tracé se fait pas à pas, c'est à dire que le stylo va d'une position à une autre de manière non continue : la table traçante reçoit une suite d'instructions, elle trace en réalité seulement des segments que l'on veut suffisamment petits pour rendre compte des courbes.

Et justement, comment connaître les tensions associées aux chiffres ? Plusieurs possibilités pour représenter les chiffres existent : soit on les affiche comme des segments pures (comme sur un afficheur 7 segments), soit on détermine les représentations paramétriques de chacun des chiffres, soit on arrondi les chiffres de la représentation 7 segments au moyen de courbes de Bézier. Nous optons pour la deuxième solution et on détermine, un peu à tâtons, les courbes paramétrées de chacun des chiffres, pour une échelle identique (nous avons simplement amélioré un peu les courbes déterminées l'année dernière). On obtient le résultat suivant (affichage droit) :



Pour le 7, on doit lever le stylo avant d'aller tracer la barre centrale.

En réalité, la grille ne sera certainement pas tout à fait droite, et l'analyse d'image aura permis de déterminer son angle d'inclinaison, c'est pourquoi nous pouvons aussi obtenir les courbes des chiffres avec un angle (exemple pour 125°) :



Nous avons maintenant la représentation normalisée des chiffres, avec ou sans angle, mais comment passe t-on à une tension ? Il suffit pour cela de connaître la taille de la case, la position de la grille et l'angle, déterminés lors du traitement d'image. Cela nous donne un offset et un facteur multiplicatif qui nous permet de déterminer les tensions associées aux chiffres à remplir. On utilise le rapport pixels/tension calculé à la partie précédente (2.4.1).

2.3.3 Envoi des données à la carte Arduino

Les tensions déterminées dans la partie d'avant par le code Python sur la Raspberry Pi 4 doivent être envoyées à la carte Arduino Mega qui se chargera ensuite de les retransmettre à la table traçante. On utilise pour se faire un port USB série.

Pour l'envoie des données depuis la Raspberry avec Python, on utilise la bibliothèque pySerial, sur le port série 'dev/ttyACM0' sous forme d'une chaîne de caractère encodée en 'utf-8' de la forme "xxxx,yyyy,z" (coordonnées suivant X et Y sur 4 chiffres et Z sur 1 chiffre). En théorie, seuls 12 bits seraient nécessaires pour coder les positions X et Y car les valeurs sont comprises entre 0 et 3300 pour X et entre 0 et 2330 pour Y. Notre moyen de transmission n'est donc pas optimal, mais au vu des délais requis par la table traçante, une transmission optimale n'apporterait rien de plus.

Du côté de la carte Arduino, nous utilisons un court programme qui reçoit simplement le signal, le décortique et renvoi des tensions à la table traçante au travers des deux CNA (X et Y) et sur le relais pilotant le lever du stylo.

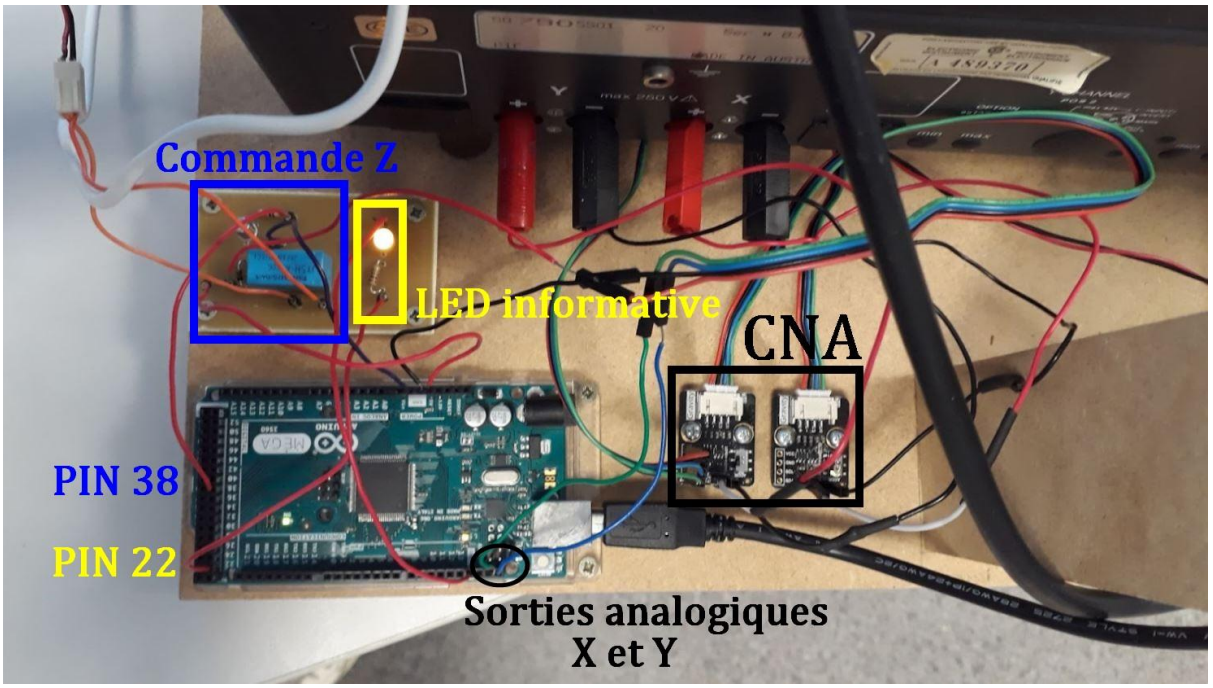
Pour atteindre les vitesses souhaitées, on transmet en 38400 Bauds, ce qui permet de gérer la vitesse dans le programme Python (en mettant le programme en pause par moment, avec la fonction *time.sleep*). On choisit volontairement une vitesse pas trop élevée afin que les chiffres soient bien écrits. Notamment, les chiffres 7 et 8 nécessitent plus de temps que les autres pour être bien écrits.

2.3.4 Communication entre l'Arduino et la table traçante

La carte Arduino est reliée à 3 ports de sortie : l'un pilote X, l'autre pilote Y et le dernier pilote Z. Pour piloter X et Y, 4 fils relient la carte au premier CNA (pour X) qui redirige une partie vers un deuxième CNA (pour Y) : un fil pour X, un pour Y, un à la masse et le dernier pour alimenter les CNA. En effet, la table traçante requiert des tensions analogiques, d'où les CNA.

Le levé du stylo est géré par la PIN 38 de la carte qui est reliée à un dispositif électronique permettant de contrôler la fermeture d'un circuit. En effet, pour que le stylo se baisse, il est nécessaire de court-circuiter deux entrées d'un port électronique de la table. Pour cela, nous utilisons un relais qui, lorsque nous lui imposons une tension, ferme le circuit en question, ce qui permet la levée du stylo.

Enfin, un dernier branchement simple sur la PIN 22 permet de contrôler l'allumage d'une LED depuis l'Arduino. Elle ne sert actuellement pas à grand-chose puisque nous n'avons pas réussi à piloter cette LED depuis la Raspberry au travers du port série (en envoyant un signal spécial…). Il serait bien plus simple, et plus logique, de connecter directement cette LED à la Raspberry pour permettre, par exemple, de changer d'état à chaque étape du système (bouton enfoncé / prise de l'image => allumée, traitement d'image => clignotante, résolution => clignotante plus rapidement, ou tout autre code).

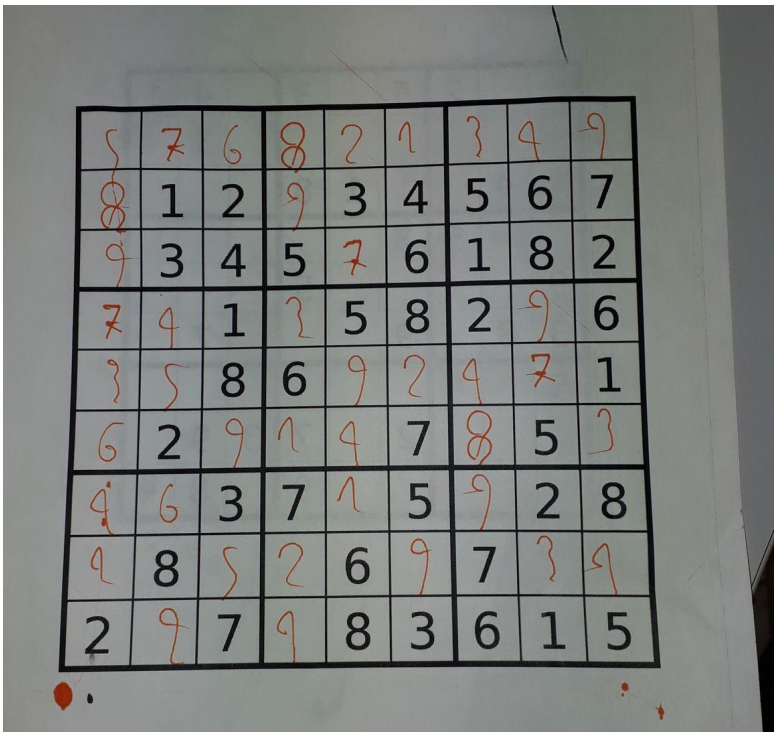


2.4 Installation des bibliothèques sur Raspberry 4

Le code développé dans la partie traitement d'image (2.1) et résolution du sudoku a été fait sur python. Cependant, celui-ci nécessitait certaines bibliothèques qui n'étaient pas présentes sur la Raspberry. Nous avons donc commencé par installer les basiques de ce langage: numpy, matplotlib, etc… Après ces installations, nous avons rencontré des gros problèmes de package pour les bibliothèques Tensor Flow et OpenCV. Ces dernières sont très volumineuses et prennent 8h chacune d'installations. Cependant, les guides de téléchargement sont très pauvres en information sur la compatibilité des versions. Nous avons donc dû réinstaller l'OS afin de vérifier la cohérence à chaque nouvelle installation. Pour la reprise du projet pour les années futures, il ne faudra pas rebooter le SSD sous peine de réeffectuer toutes ces installations. Nous avons cependant trouvé un site robuste pour l'installation [7].

2.5 Mise en route du dispositif

Pour assurer le fonctionnement de la machine, il faudra s'assurer que tous les branchements sont bien respectés et que la carte ainsi que la table soient sous-tension. Après avoir positionné le sudoku incomplet sur la table et avoir lancer le programme en question sur la Raspberry, il n'y a plus qu'à appuyer sur le bouton (relié à la Raspberry) pour laisser l'algorithme traiter le sudoku et le résoudre sur la table. On obtient ainsi un résultat similaire à celui-ci :



3 Conclusion

Au cours de ce projet, nous avons pu comprendre l'interdisciplinarité que celui-ci comportait. En effet, il est tout à fait en accord avec les matières que nous abordons lors de notre formation en SICOM. Nous avons pu nous occuper de l'acquisition de la donnée, de la traiter, de l'interpréter puis d'en tirer des résultats pour enfin les transcrire sur nos données de départ. Par ailleurs, nous avons pu expérimenter le passage d'information entre différents dispositifs. C'est surtout sur ce point là que nous avons compris la problématique de rigueur nécessaire afin que les données puissent passer de machine en machine.

Globalement nous avons réussi à améliorer les performances du système. Le traitement d'image et la résolution du sudoku sont plus rapides. Le traitement d'image est plus robuste, en particulier le modèle de reconnaissance de chiffre, et il est capable de traiter une orientation

quelconque de la grille. Pour la partie électronique, les composants ont été fixés à la structure, le montage est propre et nous avons ajouté un bouton qui permet de lancer la résolution.

Cependant, certains axes restent à améliorer :

- Pour le moment, il faut avoir un écran pour lancer le programme sur la Raspberry (avant de pouvoir appuyer sur le bouton). Pour éviter cela, il faut modifier le fichier rc.local de la Raspberry mais nous n'avons pas eu le temps.
- Il est envisageable de réaliser un circuit imprimé branchable directement sur l'Arduino., Jérémie a commencé à le faire sur KiCad mais n'a pas eu le temps de finir.
- Pour un tracé plus précis : un calcul automatique de la relation coordonnées pixels et tension à partir de points de repères sur la table traçante et une meilleure localisation des sommets de la grille (en particulier pour les grilles à bords épais).
- Un seuillage automatique de l'image de la grille qui soit davantage robuste pour éviter les erreurs de détection de chiffres.
- Une écriture sur feuille opérationnelle pour une orientation quelconque de la grille
- Accélérer l'écriture sur feuille tout en améliorant la lisibilité et qualité de l'écriture des chiffres

4 Bibliographie

Comme dans tout projet scientifique, nous nous sommes appuyés sur de nombreux documents existants, les voici :

[1] Documentation getpersPectiveTransform d'open-cv : https://theailearner.com/tag/cv2-getperspectivetransform/

[2] Réseau de neurones convolutif LeNet : https://en.wikipedia.org/wiki/LeNet

[3] Code implémentation d'un CNN LeNet : https://github.com/murtazahassan/Digits-Classification

[4] Fonctions d'activation : https://en.wikipedia.org/wiki/Activation_function

[5] Fonction Softmax : https://en.wikipedia.org/wiki/Softmax_function

[6] The Chars74K dataset : http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/

[7] Installation des bibliothèques sur Raspberry 4 : https://qengineering.eu

[8] Introduction à différentes méthodes de résolution des Sudoku : http://www-igm.univ-mlv.fr/~dr/XPOSE2013/sudoku/index.html

5 Nos codes et fichiers

Vous pouvez retrouver l'ensemble de nos codes et de nos fichiers sur le dépôt Git suivant :

https://github.com/PierreFalconnier/Projet-Sudoku.git