Rapport de stage

Université de Lorraine, Licence Informatique

Année universitaire 2023-2024

Du 22 avril 2024 au 21 juillet 2024
au Centre de recherche INRIA Nancy, équipe PIXEL

Aéna ARIA

1 Introduction

Ce stage a eu pour but le développement d'un firmware pour un robot-solveur de Rubik's Cube (cf. figure 1), écrit en Python. Une partie du stage fut en particulier consacrée à la conception d'approche de traitement de données bruitées. ("Partie recherche.")

Lors du processus de développement, l'utilisation de Python prévue initialement a posé un problème de performance. Nous avons donc choisi de plutôt utiliser Rust.

Ce stage a permis la réalisation d'un programme open source et utilisable par le grand public, dont la fiabilité est satisfaisante sur des Rubik's Cube officiels ou d'autres cubes du même type.

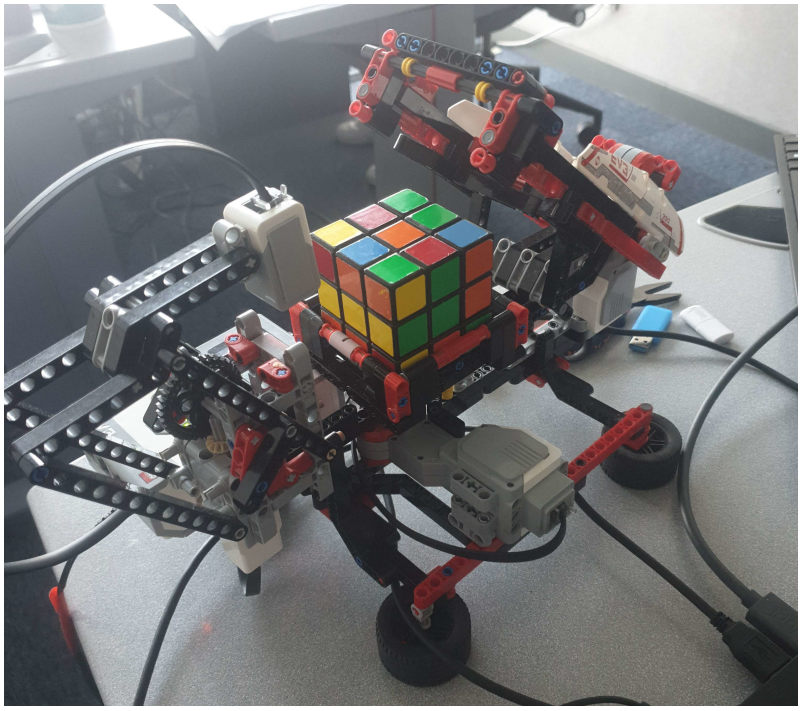

FIGURE 1 – Photographie du robot

2 Remerciements

Tout d'abord, je souhaite grandement remercier M. Dmitry Sokolov qui m'a encadré et dirigé avec beaucoup de bienveillance durant tout mon stage, et qui m'a fait confiance pour me permettre de travailler sur ce sujet passionant.

J'aimerais aussi remercier M. Nicolas Ray et M. Étienne Corman qui m'ont permis de remettre en question mes approches, et dont les conseils ont étés d'une importance capitale pour l'aboutissement du firmware, ainsi que tout le reste de l'éqipe PIXEL pour leur accueil chaleureux et l'ambiance de travail particulièrement agréable.

Merci également à Mme. Emmanuelle Deschamps et à Mme. Hélène Degeorge pour leur patience et leur aide administrative malgré les difficultés rencontrées liées à mon état civil lors de la préparation de mon stage.

Enfin, je remercie le jury correcteur de ce rapport de stage pour le temps consacré à sa lecture.

Table des matières

3 Description de l'entreprise

L'Institut national de recherche en sciences et technologies du numérique (INRIA) est un organisme public de recherche en informatique qui compte 220 équipes-projets ainsi que plus de 3800 scientifiques. Le stage s'est déroulé au centre de recherche INRIA Nancy, dans le bâtiment du LORIA qui héberge des équipes de l'INRIA, du CNRS ainsi que de l'Université de Lorraine.

Plus précisément, j'ai effectué ce stage au sein de l'équipe PIXEL qui a pour objet d'étude la géométrie dans un contexte informatique (maillages, reconstructions d'objet à partir de nuages de points 3D...). L'objet de ce stage n'est donc pas directement en lien avec les sujets habituels de l'équipe, mais certain membres avaient de l'expérience en robotique et dans d'autres domaines qui ont étés importants durant ce stage (optimisation et classification statistique).

4 Travail réalisé

4.1 Sujet et contexte

Dans une équipe de recherche incluant des enseignants-chercheurs, ce qui est le cas de l'équipe PIXEL, il arrive souvent de devoir réaliser des démonstrations au grand public afin de favoriser l'intérêt porté aux sciences informatiques, par exemple pour de potentiels futurs étudiants.
La robotique permet donc de réaliser des démonstrations très concretes et souvent intéressantes à observer, ce qui est pas toujours évident dans les domaines mathématiques et informatiques comparé à d'autres sciences.
Ainsi, un robot solveur de Rubik's Cube s'agit d'un excellent objet de démonstration, d'une part car il s'agit d'un casse-tête bien connu du grand public, et d'autre part car sa résolution est loin d'être évitente.

Pour faire des démonstrations de robotique, les robots LEGO Mindstorm s'agissent d'un excellent outil, puisqu'ils permettent un prototypage rapide ainsi qu'un assemblage très accessible, y compris à des non-spécialistes. En revanche ce matériel est assez imprécis, en particulier au niveau des capteurs, ce qui peut compromettre certaines démonstrations.

Le sujet de ce stage était donc de réaliser un logiciel embarqué compatible avec les contrôleurs LEGO Mindstorm EV3 (qui utilisent un CPU ARM cadencé à 300 MHz, donc avec une puissance de calcul faible) qui permettrait la résolution d'un rubik's cube sur un matériel similaire à ceux des robots Mindcuber [1].

4.2 Objectifs

le logiciel utilisé pour les robots Mindcuber a en particulier quelques problèmes qu'on souhaiterait corriger dans notre version, le plus problématique étant que le taux de succès de résolution du cube était très faible et dépendant de la luminosité ambiante.

Idéalement, ce nouveau firmware doit donc pouvoir fonctionner dans des faibles conditions de luminosités ainsi qu'avec des cubes non-officiels qui comportent des faces de couleurs différentes de celles du cube original. Puisqu'il s'agit d'un robot destiné à des démonstrations, il est également important que le temps de calcul de solution soit faible afin ne pas faire patienter un observateur.

4.3 Méthodologie

Puisque le stage concerne du développement sur système embarqué, l'environment de développement ainsi que les technologies choisies peuvent avoir un impact important, à la fois sur les performances du programme mais aussi sur sa facilité d'écriture. Voici les différentes technologies utilisées :

- **Langage de programmation :** Rust [2] est un langage compilé et qui permet d'assurer la sécurité mémoire. Sa rapidité d'exécution est proche de celle du C, et ce langage hérite de plusieurs paradigmes inspirés d'autres langages (Impératif, fonctionel, orienté objet...). C'est un choix pertinent pour rapidement prototyper sans perdre en performance, ce qui est particulièrement important sur un système embarqué. Rust permet également la cross-compilation, c'est à dire qu'il est possible de compiler pour un processeur ARM même depuis un PC d'une autre architecture. Il est donc possible de compiler le programme directement depuis le PC de développement et ainsi gagner du temps comparé à s'il fallait compiler directement sur le contrôleur EV3
- **Système d'exploitation du système embarqué :** L'OS par défaut dans le EV3 étant prévu pour être utilisé avec le logiciel officiel LEGO qui est limité, il est nécessaire de remplacer le système d'exploitation du contrôleur. ev3dev [3] est un projet qui permet d'avoir un système linux complet basé sur debian fonctionnant sur un contrôleur EV3. De plus, cet OS permet d'effectuer des appels systèmes aux périphériques Lego Mindstorm (moteurs, capteurs) directement en Rust avec le binding ev3dev-lang-rust [4].
- **Éditeur de code :** Visual Studio Code possède plusieurs extensions qui facilitent le développement avec Rust et ev3dev.

Le stage ne portant pas spécifiquement sur la recherche de la solution au cube scanné, le choix d'un algorithme de résolution du cube devient très important pour répondre aux contraintes de rapidité de calcul. Nous avons choisi d'utiliser l'algorithme de Kociemba [5], implémenté en rust avec le package Kewb [6]. Cet algorithme ne trouve pas nécessairement une solution optimale, c'est à dire en un minimum de mouvements du cube, mais il permet un temps d'exécution faible (moins de 5 secondes pour trouver une solution sur le contrôleur).

En ce qui concerne le matériel du robot, il possède un capteur de couleur, et trois éléments mobiles : un bras pour placer le capteur de couleur au dessus du cube et le retirer lors de l'application de mouvements, une base sur laquelle le cube est placée qui permet de lui appliquer un rotation, et un deuxième bras qui "atrappe" le cube, soit pour le maintenir en place lors de la rotation d'une face, soit pour retourner le cube et exposer une nouvelle face au capteur.

4.4 Déroulement du stage

Les premières semaines du stage consistaient à écrire le code pour l'aspect mécanique du robot : mouvements des moteurs, procédure de scan du cube, application d'une solution...

Une fois que la partie mécanique était suffisamment au point, nous pouvions récupérer un tuple de couleur RGB pour chacune des facettes du cube. Afin de transformer ces tuples

RGB en une des 6 couleurs du cube pour déterminer l'état du cube inséré dans le robot, nous avons d'abord commencé par effecturer une classification naïve. Ainsi, le premier cube fut résolu le 13 mai.

Cependant, nous nous sommes rendus compte qu'une simple classification ne peut pas suffir à obtenir des résultats fiables : les données récupérées par le capteur de couleur étant fortement bruitées. Le reste du stage fut donc consacré à l'élaboration d'un algorithme permettant de déduire l'état du cube à partir des données bruitées en entrée.

Les deux dernières semaines de stage ont permis d'affiner le logiciel pour l'utilisateur final ainsi que de rédiger un rapport technique pouvant servir de base pour une publication future des résultats obtenus.

4.5 Résultats

Grâce aux recherches effectuées durant ce stage, nous avons pu parvenir à l'élaboration d'un algorithme qui permet de retrouver l'état réel du cube scanné par le robot.

Données d'entrées

```
             |************|
             |*U1**U2**U3*|
             |************|
             |*U4**U5**U6*|
             |************|
             |*U7**U8**U9*|
             |************|
|************|************|************|************|
|*L1**L2**L3*|*F1**F2**F3*|*R1**R2**R3*|*B1**B2**B3*|
|************|************|************|************|
|*L4**L5**L6*|*F4**F5**F6*|*R4**R5**R6*|*B4**B5**B6*|
|************|************|************|************|
|*L7**L8**L9*|*F7**F8**F9*|*R7**R8**R9*|*B7**B8**B9*|
|************|************|************|************|
             |************|
             |*D1**D2**D3*|
             |************|
             |*D4**D5**D6*|
             |************|
             |*D7**D8**D9*|
             |************|
```

Figure 2 – Patron des facettes du cube

Pour scanner le cube, le robot utilise le capteur de couleur qui renvoie un tuple RGB unique correspondant à la couleur de la surface en dessous. Ainsi, nous pouvons récupérer un tuple de 54 vecteurs P, avec chaque tuple correspondant à une facette du cube à une position connue d'avance (cf. figure 2).

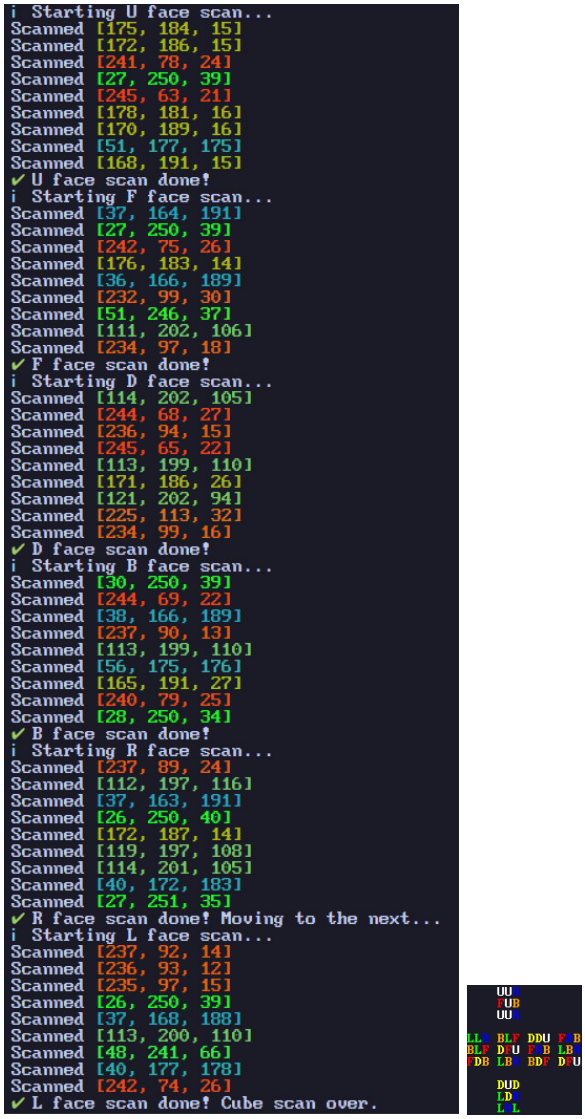

FIGURE 3 – Capture d'écran du terminal affichant les tuples scannés par le robot (à gauche) et le résultat attendu pour ce scan (à droite)

Étant donné qu'il y a 6 faces différentes sur un cube, nous devrions pouvoir observer 6 différentes couleurs. Notre objectif est donc alors de classifier chacun de ces 54 tuples en 6 couleurs de 9 tuples chacunes(cf. figure 3).

À cause de l'imprécision du matériel et des conditions extérieures (comme la luminosité de l'environment du robot), les données récupérées sont fortement bruitées (cf. figure 4). Heureusement, il existe différentes contraintes nous permettant de trouver la bonne classification qui correspond à l'état réel du cube

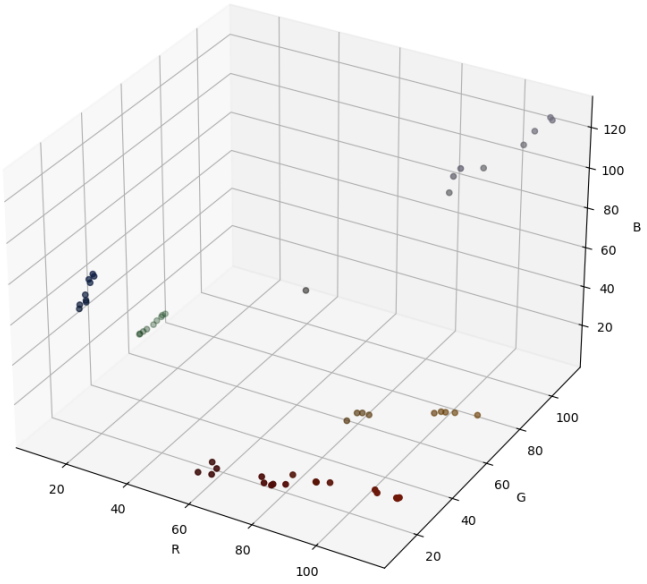

FIGURE 4 – Nuage de point 3D contenant les tuples RGB scannés sur un cube officiel

Formalisation du problème

Nous obtenons donc une matrice P, qui contient les tuples RGB scannés :

$$P = \begin{bmatrix} p_1 \\ ... \\ p_{54} \end{bmatrix}$$

Nous souhaitons trouver une matrice C de dimension 54×6, pour laquelle $C_{i,j} = 1$ si p_i fait partie de la classe (couleur) j, sinon $C_{i,j} = 0$. Par exemple, si p_1 faisait partie de la première classe, p_2 de la seconde et p_3 de la quatrième, nous devrions obtenir :

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ ... & & & & & \end{bmatrix}$$

Nous définissons aussi M la matrice contenant la moyenne des tuples de toute les classes :

$$M = \frac{C^T P}{9}$$

Nous pouvons juger à quel point la classification est fidèle au scan en effectuant la somme des distances entre les tuples et la moyenne de leur classe.

Ainsi, nous recherchons un C qui minimise $||P - CM||$ tel que C représente un état du cube qui peut être résolu.

Pour déterminer si un cube donné est soluble ou non, nous faisons appel à l'algorithme de Kociemba. Cependant, nous pouvons déterminer deux contraintes majeures de solubilité qui sont nécessaires mais non suffisantes :

— Une facette ne peut pas être de deux couleurs différentes, donc chaque ligne de C doit contenir exactement un seul 1
— Il y a exactement neuf facettes de la même couleur, donc chaque colonne de C doit contenir neuf 1

Déroulement général de l'algorithme

Notre algorithme fonctionne en trois étapes distinctes afin de résoudre ce problème d'optimisation.

— Lors de l'étape une, nous recherchons un C qui valide nos deux conditions nécessaires et suffisantes et qui est une approximation de la classification recherchée.
— Lors de l'étape deux, nous modifions le C de l'étape une pour obtenir une meilleure minimisation de $||P - CM||$ sous nos deux contraintes.
— Lors de l'étape trois, nous cherchons à satisfaire la contrainte de solvabilité sur le C trouvé lors de l'étape deux.

Implémentation

Étape une Nous considérons les facettes du centre (U5,F5,L5,R5,B5,D5) comme membres de chacune des différentes classes, étant donné qu'il n'y a pas deux différentes facettes du centre qui sont de la même couleur. Ensuite, nous calculons les distances euclidiennes de chacune des facettes non-centres à chacun des centres.

Ceci nous permet donc d'itérer sur les facettes triées par leur plus petite distance à un centre, et nous attribuons chacune de ces facettes à la classe du centre le plus proche qui n'est pas déjà pleine (neuf facettes en prenant en compte le centre).

Cette méthode nous permet de satisfaire les deux conditions nécessaires, puisque nous attribuons chaque facette à une unique classe et nous ne pouvons pas attribuer une facette à une classe qui a plus de 9 éléments. Cependant, dans certain cas, il est possible d'obtenir des classifications abérrantes, ce qui explique le besoin de l'étape deux, où ces classifications invalides seront corrigées.

Dans notre implémentation, nous avons trouvé que traiter séparément les facelettes des coins et des arêtes comme classes séparées de 5 éléments (1 centre et 4 facettes de coins ou arêtes) et ensuite combiner les résultats permettait de réduire le nombre de classification abérrantes. De plus, cette technique permet d'effectuer une optimisation plus tard lors de l'étape trois.

Étape deux Nous définissons une opération qui permet de modifier C en préservant la cardinalité des classes : l'échange de lignes de C (un *swap*). Ainsi, nous pouvons swap les lignes qui réduisent le plus $||P - CM||$ jusqu'à ce que plus aucun swap ne permette une réduction.

Étape trois Nous pourrions explorer tous les swaps de manière exhaustive afin de trouver le C le plus proche qui satisfait notre contrainte de solubilité. Mais en prenant en compte qu'il y a 54 lignes dans C, cela amènerait à $(54 \times 53)^k$ possibilités pour k swaps consécutifs. Dans certain cas, jusqu'à 8 swaps consécutifs sont nécessaires afin de retrouver une configuration solvable, ce qui rend cette approche inadéquate en raison du temps de calcul nécessaire qu'elle impliquerait.

Alors il devient nécessaire de réduire le nombre de possibilités à explorer :

- Nous pouvons retirer tous les swaps impliquant une facette du centre. Puisque les classes ont étés construites sur la base de ces centres, on peut être sûrs de leur affectation.
- Si les facettes des coins et des arêtes ont été classifiées séparément lors de l'étape une, alors nous pouvons aussi retirer tous les swaps entre une arête et un coin, puisqu'une telle opération modifierait la cardinalité des deux types.
- L'ordre des swaps consécutifs peut être ignoré, car nous avons constaté que toute les possibilités sont explorées à terme, avec assez de swaps consécutifs.
- Les swaps impliquant des tuples qui sont très loin l'un de l'autre, et qui modifieraient alors grandement $||P - CM||$ peuvent être retirés, au risque de perdre la solution.

Cette dernière optimisation est très importante car le nombre de tuples retirés peut être choisi afin d'obtenir plus de performance au risque de perdre un swap qui serait nécessaire afin de trouver la solution. Dans des conditions réelles, nous avons pu voir que pour k swaps consécutifs sur un ensemble de S swaps possibles, garder uniquement les $|S|/2^{k-1}$ meilleurs swaps permettait d'obtenir une performance acceptable, même sur le processeur limité du contrôleur EV3.

Ainsi, en explorant cet ensemble limité de swaps, il est alors possible de facilement trouver un C qui satisfasse nos contraintes. Il ne s'agit en revanche pas nécessairement de la solution cherchée.

Pour notre implémentation, nous avons choisi d'arrêter l'exploration des possibilités lorsque la taille de l'ensemble limité est plus petit que le nombre de swaps consécutifs, et nous gardons le C qui donne $||P - CM||$ minimal parmis tous les C valides trouvés.

Performance de l'algorithme

Cet algorithme a été testé sur un ensemble de 35 scans en conditions réelles en luminosité faible d'un cube résolu (où toute les faces sont de couleur unies).
L'application de l'algorithme résulte en 35 configurations correspondantes à celle attendue, tout en restant sous les 5 secondes de temps d'exécution.

Des tests plus aboutis, y compris sur des cubes non-officiels avec des couleurs différentes de celles du cube officiel, ont aussi aboutis par un succès. Seulement un seul cube parmis les 5 cubes différents testés n'a pas pu être résolu par le robot, à cause de données inexploitables fournies par le capteur de couleur.

4.6 Continuité du travail réalisé

Le code fut publié en tant que logiciel libre et est disponible sur la plateforme Github à l'adresse `https://github.com/Seliaste/mindsolver` sous licence Apache 2.0.
Des binaires ainsi que des instructions d'utilisations sont également disponibles afin de permettre à quiconque d'utiliser le logiciel sur son propre robot.

Une publication est, au moment où ce rapport est rédigé, en cours de rédaction par les membres de l'équipe afin de communiquer sur les résultats obtenus.

5 Conclusion

Pour conclure, au cours de ces 3 mois de stage, nous avons pu atteindre tous les objectifs fixés. Le logiciel est fonctionel et fiable, et le robot est désormais prêt à pouvoir être utilisé pour réaliser des démonstrations.

Ce stage m'a aussi permis de solidifier mes acquis au cours de ma formation, en particulier dans les domaines d'optimisation linéaire, de programmation fonctionnelle ou encore de probabilités et statistiques. Le sujet était très intéressant et je trouve qu'il s'agit d'une bonne synthèse des compétences obtenues au cours de ma licence.

J'ai le sentiment d'avoir beaucoup appris grâce aux chercheurs de l'équipe sur des sujets totalement indépendants de mon stage et plus en rapport avec le domaine habituel de l'équipe. Pour moi, il est sûr que cette découverte du domaine de la recherche m'aidera grandement dans les décisions que je devrai prendre au cours de la suite de mon parcours d'étude.

Références

[1] David Gilday, *Robot Mindcuber* https://mindcuber.com/

[2] *Langage Rust* https://www.rust-lang.org/

[3] *Système d'exploitation ev3dev* https://www.ev3dev.org/

[4] *Bindings ev3dev rust* https://github.com/pixix4/ev3dev-lang-rust

[5] Herbert Kociemba, *The Two-Phase Algorithm* https://kociemba.org/cube.htm

[6] Luckas Ranarison, *Librairie Kewb* https://github.com/luckasRanarison/kewb